Diseño y evaluación de un clúster HPC: Software de sistema

Autor:
Cristobal Ortega Carrasco

Grado en Ingeniería Informática
Especialidad en Ingeniería de Computadores

26 de Junio de 2014

Director:
David López Álvarez, Arquitectura de computadores

Facultad de Informática de Barcelona
Universidad Politécnica de Cataluña (UPC) - BarcelonaTech

Resumen

Este trabajo trata de qué software a nivel de sistema es necesario para poder crear un clúster para fines de High Perfomance Computing (HPC). Para ello se realiza un state of the art del software existente y más conocido. También se trata el cómo instalarlo y configurarlo de la manera más óptima para tener un mejor resultado, no se llega a optimizar a nivel de kernel. Para esta parte del trabajo, se dispone de un pequeño clúster de procesadores de bajo consumo ARM para experimentar con el distinto software, esto supone encontrarse con problemas que quizá con otra plataforma más típica no ocurrirían.

El trabajo tiene como objetivo final el dejar un clúster funcional a nivel de software de sistema, para poder correr aplicaciones de HPC sobre él y poder obtener métricas de rendimiento.

Resum

Aquest treball és sobre quin tipus de software a un nivell de sistema és necessari per poder crear un clúster amb una finalitat per HPC. Per això, es realitza un state of the art del software existent. També és sobre com instal·lar i configurar aquest software per que corri de manera més òptima per arribar a tenir un millor resultat, a nivell de kernel no es fan optimitzacions. Per aquest part del treball, es disposa d'un clúster de processadors de baix consum ARM per experimentar amb el diferent software, això podria implicar trobar-se mes problemes dels que hi podríem trobar si utilitzéssim una plataforma més típica.

El treball té com a objectiu final deixar un clúster funcional preparat a nivell de software de sistema, per córrer diverses aplicacions HPC i poder obtenir el seu rendiment.

Abstract

This project is about what kind of system software is needed to be able to make a HPC cluster. In order to achieve this, a state of art is made about system software used nowadays. It is also about how install,configure and optimize this software to get the best results, but no optimizations are made at the kernel level. For this part of the work, there is a cluster of low-power ARM processors to experiment with different software, this could mean finding some problems that it might not occur if another platform more typical would have been used.

The goal of this project is to get a functional cluster prepared with system software capable of running HPC applications and get its performance.

Índice general

Resumen **2**

Prefacio **8**

1. Introducción **10**
1.1. Orígenes . . . 10
1.2. Problemática . . . 10
1.3. Student Cluster Challenge . . . 11
1.4. Objetivos . . . 11
1.4.1. Objetivos Generales . . . 11
1.4.2. Objetivos individuales . . . 12

2. Planificación y presupuestos **14**
2.1. Planificación . . . 14
2.1.1. Estudio Teórico . . . 14
2.1.2. Aplicación Práctica . . . 14
2.2. Estimación temporal . . . 15
2.2.1. Listado de Tareas . . . 16
2.2.2. Diagrama de Gantt . . . 17
2.2.3. Recursos . . . 18
2.3. Presupuesto . . . 18
2.3.1. Recursos humanos . . . 19
2.3.2. Hardware . . . 19
2.3.3. Software . . . 20
2.3.4. Gastos generales . . . 20
2.3.5. Presupuesto total . . . 21

3. State of art **22**
3.1. Stack de software en High Perfomance Computing . . . 22
3.1.1. HP . . . 22
3.1.2. IBM . . . 23
3.1.3. Cray Inc. . . . 24
3.1.4. SGI . . . 25
3.2. Sistemas operativos . . . 25
3.3. Monitorización . . . 26
3.3.1. Monitorización de red . . . 27
3.3.2. Monitorización de sistema . . . 27
3.4. Software de desarrollo . . . 28

3.4.1. Compiladores . . . 28
3.4.2. Profiling . . . 30
3.5. Software de scheduling . . . 30
3.6. Message Passing Interface . . . 31
3.6.1. Implementaciones MPI . . . 33
3.7. Librerías . . . 33
3.7.1. ATLAS . . . 34
3.7.2. FFTW . . . 35

4. Diseño de un clúster **36**
4.1. Clúster a usar . . . 36
4.2. Selección de software . . . 39
4.3. Sistema operativo . . . 39
4.4. Monitorización . . . 42
4.4.1. Ganglia . . . 42
4.5. Software de desarrollo . . . 46
4.5.1. GCC . . . 46
4.5.2. Extrae . . . 46
4.6. Message Passing Interface . . . 48
4.6.1. OpenMPI . . . 48
4.6.2. MPICH . . . 49
4.6.3. Evaluación . . . 49
4.7. Librerías . . . 51
4.7.1. ATLAS . . . 51
4.7.2. FFTW . . . 54
4.8. Problemas con los nodos . . . 56

5. Conclusiones **58**
5.1. Conclusiones . . . 58
5.1.1. Objetivos . . . 58
5.1.2. Conclusión personal . . . 58
5.2. Trabajo futuro . . . 59

6. Conclusiones de equipo **60**
6.1. Conclusiones de cara a la competición . . . 60
6.2. Trabajo futuro . . . 61

Agradecimientos **62**

Apéndices **63**

A. Script: cambiar hostname **64**

B. Script: copiar clave pública **65**

C. Script: ssh a nodos **66**

Índice de cuadros

2.1. Distribución de horas de trabajo según rol. 15
2.2. Costes asociados a recursos humanos. 19
2.3. Costes derivados de la compra de Hardware. 19
2.4. Desglose de los gastos generales. 21
2.5. Resumen presupuesto total. 21

3.1. Software stack de Triolith (14) 22
3.2. Software stack de IBM (7) 23
3.3. Software stack de Tianhe-2 (12) 25

4.1. Stack de software a usar 39
4.2. Tiempos para precisión simple de ATLAS 53
4.3. Tiempos para precisión doble de ATLAS 53

Índice de figuras

2.1. Listado de Tareas 16
2.2. Diagrama de Gantt 17
2.3. Estado del Laboratorio C6-103 al llegar. 20

3.1. Stack de software usado por HP (18) 23
3.2. Stack de software usado por IBM (7) 24
3.3. Stack de software usado por Cray Inc. (21) 24
3.4. Evolución de los Sistema operativo (SO) en el TOP500 25
3.5. Mercado de Sistemas Operativos en el TOP500 26
3.6. Perfomance de Intel compiler collection (23) 29
3.7. Comparación entre Automatically Tuned Linear Algebra Software (ATLAS) y MKL (5) 34
3.8. Comparación entre ATLAS y MKL (5) 34
3.9. Comparación entre Fastest Fourier Transform in the West (FFTW) y MKL (5) . 35

4.1. Vista frontal del clúster (cerveza para comparar la escala) 37
4.2. Vista lateral del clúster (cerveza para comparar la escala) 37
4.3. Vista cercana del clúster 38
4.4. Vista del *switch* de interconexión 38
4.5. Web principal de Ganglia 45
4.6. Información del nodo 86 45
4.7. Prueba de latencia entre OpenMPI y MPICH 50
4.8. Prueba de ancho de banda entre OpenMPI y MPICH 51
4.9. Prueba de rendimiento de ATLAS con doble precisión 54
4.10. Prueba de rendimiento de FFTW con precisión simple 55
4.11. Prueba de rendimiento de FFTW con precisión doble 56

Prefacio

Este trabajo forma parte de un proyecto colaborativo de investigación entre estudiantes que se ubica en el área de la informática conocida como HPC. Comparte título y se complementa con el trabajo de tres compañeros de proyecto: David Trilla en el ámbito del hardware, Cristobal Ortega en el de software de sistema y Constantino Gómez por el apartado de aplicaciones. Por tanto, los siguientes capítulos son compartidos a nivel de proyecto: este prefacio, introducción, planificación y presupuestos, sostenibilidad y por último conclusiones de equipo. Estos capítulos son compartidos porque aportan información esencial común a todo el proyecto, así como una descripción del trabajo conjunto que ha realizado el equipo.

Hoy en día el HPC es una de las herramientas principales para el desarrollo de la ciencia. Por norma general las aplicaciones HPC comparten una característica común, se pueden desmenuzar en subtareas para poder así ejecutarse de manera paralela en gran cantidad de procesadores.

El principal exponente de este tipo de investigación son los supercomputadores, máquinas con capacidades de cómputo muy por encima de la mayoría de ordenadores, y que son imprescindibles para casi cualquier campo científico e investigación. El HPC se encarga de que estas máquinas sigan evolucionando para permitir que la ciencia siga avanzando a pasos cada vez mayores.

Una de las conferencias más importantes en materia HPC es la International Supercomputing Conference (ISC). Los asistentes, mayoritariamente profesionales del sector, participan en charlas técnicas, conferencias y talleres entre otros. Para este trabajo, no es el evento principal el que resulta de interés, sino la competición HPC: Student Cluster Competition (SCC). En esta competición, que participan universidades de todo el mundo, equipos de estudiantes compiten por conseguir los mejores resultados de rendimiento.

La creación del grupo responde a dos necesidades: la de dar cabida a los tres aspectos técnicos más importantes de la tecnología HPC en un proyecto común, y segundo, la de formar un equipo y las opciones que se tendrían para ganar una competición como la Student Cluster.

Capítulo 1

Introducción

1.1. Orígenes

A principios de septiembre de 2013, Álex Ramírez reúne a 7 estudiantes de la especialidad de ingeniería de computadores, entre ellos los 3 integrantes del grupo, y nos propone formar un equipo con intención de participar en la ISC '14 que se celebra en Leipzig del 21 al 25 de Junio en 2014

A lo largo de septiembre y octubre se estudian los requerimientos de la competición y elaboramos el documento de inscripción con información de los participantes y un primer planteamiento del clúster.

A principios de diciembre la organización nos comunica que no hemos sido admitidos en la competición sin aportar mayor explicación.

En enero de 2014 acordamos seguir con el proyecto a modo de TFG pese a no estar admitido. El grupo se reduce a 3 personas: Constan Gómez, David Trilla y Cristobal Ortega.

1.2. Problemática

Por un lado la problemática que se plantea es la siguiente: en caso de querer competir en el futuro en una competición como el SCC, qué competencias de equipo y conocimientos técnicos deberíamos desarrollar.

Por otro lado, nos encontramos otro problema, más general y de más alcance, como es el consumo de los supercomputadores hoy en día. Actualmente, el mantenimiento de los centros de computación tiene un coste muy elevado. Gran parte del coste se debe al consumo eléctrico de sus equipos y la refrigeración necesaria para mantenerlos funcionando. Este problema es atacado de forma indirecta en el SCC por la limitación de consumo a 3.000 Vatios (W), y en el clúster que construiremos con *hardware* de bajo consumo.

Para ayudarnos a definir los objetivos del trabajo, veamos primero en detalle en que consiste la competición.

1.3. Student Cluster Challenge

El SCC es una competición que se celebra en el ámbito del ISC, el escaparate sobre HPC en Europa. El evento ISC '14 tiene lugar entre los días 21 y 25 de Junio en Leipzig Alemania. Esta competición será nuestro marco de referencia, dirigiendo nuestro proyecto como si fuéramos a participar en ella, usando sus directrices y parámetros. El evento consiste en reunir a varios equipos, de hasta 6 estudiantes sin graduar, que junto con un pequeño sistema para supercomputación y a lo largo de 4 días, compitan entre ellos en diferentes categorías para determinar el sistema ganador, mediante el montaje y la optimización de los parámetros de las aplicaciones.

Existen 3 categorías a las cuales se opta a premio, la de mejor resultado con High-Performance Linpack (HPL), que es la versión para HPC del *software* de benchmarking LINPACK, la de votación a favorito, y la categoría general donde cuenta la puntuación total de todos los benchmarks.

La característica de la competición de premiar al eficiencia da lugar a la principal restricción que se impone en la competición, que limita el "presupuesto para el consumo". Esto limita el consumo de cualquier clúster en competición, incluyendo sus elementos para interconexión, a un total de 3.000W. (10)

Debido a todas las anteriores normas, nuestra intención es diseñar dos clústeres pensados para presentar al concurso. Un clúster teórico, que cumpla los requisitos anteriores, y que nos permitirá liberarnos de la necesidad de depender del coste de su creación, y posteriormente, con el *hardware* disponible, crear un clúster que pueda ejecutar eficientemente los benchmarks del concurso, con especial atención en el HPL, y que también cumpla la restricción de consumo.

1.4. Objetivos

A continuación se desglosan los objetivos generales del proyecto y los objetivos individuales de cada uno de los tres trabajos.

1.4.1. Objetivos Generales

Basándonos en los requerimientos vistos en la sección anterior se establecen los siguientes objetivos generales y su justificación. Hemos dividido el proyecto en dos fases.

Objetivos de la primera fase

- **Estudio del estado del arte HPC**
- **Diseñar y montar un clúster con un consumo inferior a los 3kW.**

Para poder diseñar adecuadamente un clúster necesitamos realizar previamente un estudio del estado del arte. De esta manera podremos extraer cuales son los elementos indispensables para su montaje, y además, adquirir otros conocimientos relacionados que nos ayudarán durante

todo el desarrollo del proyecto.

El segundo objetivo se aborda de dos maneras diferentes. Por un lado se realiza el diseño teórico de un clúster con procesadores y aceleradores convencionales. Por otro, el diseño y **montaje** de un prototipo de clúster con procesadores móviles de bajo consumo.

Dada la necesidad de asistir con un clúster propio a la competición, es necesario trabajar de primera mano con el hardware y el software de sistema para ser más competitivos.

Objetivos de la segunda fase

- **Evaluación del clúster basado en procesadores de bajo consumo.**

En este momento, mediante benchmarks y aplicaciones, realizaremos las pruebas empíricas que demostrarán que nuestra solución al problema está a la altura de la competición. En caso de que no estarlo, se buscará poder señalar de manera precisa la causa de las deficiencias de rendimiento (cuellos de botella) ya sean de diseño, de configuración o de uso incorrecto de las herramientas de evaluación.

Dado que ningún integrante del grupo ha participado en algún proyecto similar anteriormente, el proceso de evaluación servirá también para el desarrollo de las habilidades de equipo necesarias en la competición, ya que, gran parte de las técnicas y los test aplicados con el fin de optimizar la máquina y las aplicaciones de este trabajo son extrapolables a otras configuraciones de hardware y otros programas.

1.4.2. Objetivos individuales

Objetivos de la sección de hardware

Fase 1

- Investigación sobre el estado del arte del *hardware* en el mundo de HPC, cuáles son las tecnologías más usadas y los componentes necesarios para construir un clúster.
- Crear un clúster para HPC de manera conceptual para poder competir en el SCC, sin limitación económica.
- Evaluar la tecnología usada en el SCC y las capacidades del clúster teórico diseñado.

Fase 2

- Analizar el *hardware* a nuestro alcance, disponible para construir un clúster para HPC.
- Montar y configurar el *hardware* para tener una plataforma funcional con múltiples nodos sobre la que poder ejecutar *software*
- Evaluar el rendimiento del *hardware*, las diferencias entre los resultados esperados y los reales, y comparar con el clúster conceptual de la primera fase y otros sistemas con tecnologías distintas.

Objetivos de la sección de software de sistema

Fase 1

- Investigar que software de sistema es necesario habitualmente para correr aplicaciones del tipo HPC.
- Estudiar el estado actual en los sistemas de supercomputación para saber que stack de software usan.
- Seleccionar el software de sistema que necesitemos y elegir el que mejor se nos adapte a nosotros, ya sea por compatibilidad con el hardware o aplicaciones a correr, por documentación existente o requisitos diversos.

Fase 2

- Basado en la fase 1, instalar y configurar todo el stack de software para crear un clúster totalmente funcional. Es posible que haya que seleccionar otro tipo de software por la plataforma usada.
- Experimentar con distintas versiones de paso de mensajes MPI para saber realmente cuál se adapta a nuestro sistema

Objetivos de la sección de aplicaciones

Fase 1

- Investigar las aplicaciones en el estado del arte actual y analizar las más relevantes para nuestro proyecto.
- Averiguar que opciones existen de cara a optimizar las aplicaciones que se ejecutarán en el clúster.

Fase 2

- Evaluar el rendimiento de las aplicaciones descritas a nivel de nodo y a nivel de clúster.

Capítulo 2

Planificación y presupuestos

2.1. Planificación

2.1.1. Estudio Teórico

La primera fase, es de investigación activa, sobre HPC y el SCC, e información sobre todo lo necesario para la instalación y preparación de un clúster. Esto incluye *hardware*, *software* de sistema y aplicaciones. Esta parte del proyecto recabará la información necesaria para elaborar un clúster teórico con el que ir a competir en el SCC.

No se esperan grandes contratiempos durante esta fase. Los problemas que puedan surgir serán derivados de la poca información disponible sobre algunos componentes del clúster.

2.1.2. Aplicación Práctica

La segunda fase, basada en la experimentación, es en la cual usamos la información recogida en la fase anterior para crear un clúster totalmente funcional. En esta fase es donde es más probable que surjan dificultades técnicas, ya que al ser un mundo casi nuevo, como por ejemplo, que la información recogida en la fase anterior no se ajuste a nuestra arquitectura o *software* escogido.

Los posibles retrasos de ponernos a montar el clúster, instalación de software y benchmarks deberemos tenerlos en cuenta y aplazar el resto de tareas que tengamos, como es la optimización de software y aplicaciones, esto implica que obtendremos peores rendimientos de los que realmente podríamos conseguir. Ya que para poder obtener las métricas de rendimiento necesitamos un clúster funcionando. Y aunque no sea un clúster optimizado todo lo posible, el objetivo de tener un clúster de HPC estará conseguido. Los posibles retrasos que aparezcan en esta sección puede aparecer de errores e incompatibilidades en la fase de instalación del clúster, el tiempo adicional será recortado de las tareas más opcionales dispuestas al final del proyecto, correspondientes a la parte de optimización, lo que implicará que obtendremos peores rendimientos de los que realmente podemos conseguir, ya que la instalación y puesta en marcha del clúster es esencial, sin embargo el proyecto estará finalizado ya que tendremos un clúster totalmente funcional.

2.2. Estimación temporal

La temporización en este proyecto toma especial importancia por dos motivos: hay un gran volumen de tareas a realizar que requieren un esfuerzo conjunto y la alta incertidumbre que albergan algunas de ellas. Debido a las fuertes dependencias entre tareas es imprescindible tener en mente las fechas comprometidas para garantizar que todo el grupo cumple con sus objetivos.

Desde el principio se acuerdan unas horas fijas de dedicación semanal en grupo. Por un lado, nos ayudará a empezar con buen ritmo con la parte teórica y a funcionar como un equipo, y por otro, tener margen de reacción de cara al final del proyecto donde se prevén más problemas.

Tarea	**Dedicación por persona**
Estudio previo	125h
Montaje y configuración	155h
Benchmarking	35h
Análisis de resultados	60h

Cuadro 2.1: Distribución de horas de trabajo según rol.

2.2.1. Listado de Tareas

		Name	Duration	Start	Finish	Predeces	Resources
1		Investigación componentes hardware/software de un cluster y Sta	10d	21/02/2014	06/03/2014		
2		**Estudio por componentes**	**20d**	**07/03/2014**	**03/04/2014**	**1**	**David Trilla**
3		Estudio fuente de alimentación	3d	07/03/2014	11/03/2014		
4		Estudio CPUs	3d	11/03/2014	13/03/2014		
5		Estudio Aceleradores	3d	13/03/2014	17/03/2014		
6		Estudio Memorias	5d	17/03/2014	21/03/2014		
7		Estudio Almacenamiento	5d	24/03/2014	28/03/2014		
8		Estudio de redes de interconexión	5d	28/03/2014	03/04/2014		
9		**Estudio por plataformas**	**10d**	**07/03/2014**	**20/03/2014**	**1**	**Constantino Gómez**
10		Sistemas homogéneos	5d	07/03/2014	13/03/2014		
11		Sistemas heterogéneos	5d	10/03/2014	14/03/2014		
12		Sistemas de bajo consumo: bluegene	5d	14/03/2014	20/03/2014		
13		**Estudio por componentes (software)**	**21d**	**07/03/2014**	**04/04/2014**	**1**	**Cristóbal Ortega**
14		Estudio SOs	3d	07/03/2014	11/03/2014		
15		Estudio drivers	5d	11/03/2014	17/03/2014		
16		Estudio compiladores	3d	17/03/2014	19/03/2014		
17		Estudio librerías	3d	19/03/2014	21/03/2014		
18		Estudio runtime	5d	21/03/2014	27/03/2014		
19		Estudio gestor de colas	5d	27/03/2014	02/04/2014		
20		Estudio monitorizadores	3d	02/04/2014	04/04/2014		
21		**Estudio de aplicaciones**	**12d**	**07/03/2014**	**24/03/2014**	**1**	**Constantino Gómez**
22		Benchmarks de Memoria	3d	07/03/2014	11/03/2014		
23		Benchmarks coma flotante / enteros	4d	11/03/2014	14/03/2014		
24		Benchmarks de red	3d	14/03/2014	18/03/2014		
25		Aplicaciones reales	5d	18/03/2014	24/03/2014		
26		**Instalación de un cluster HPC**	**28d**	**07/04/2014**	**14/05/2014**	**2,9,13,21**	**David Trilla,Cristóbal Orte**
27		Cuestiones prácticas	7d	07/04/2014	15/04/2014		
28		Selección del hardware	3d	10/04/2014	14/04/2014		
29		Selección del software	3d	14/04/2014	16/04/2014		
30		Instalación del hardware	7d	16/04/2014	24/04/2014		
31		Instalación del software	15d	24/04/2014	14/05/2014		
32		**Aplicaciones HPC**	**11d**	**15/05/2014**	**29/05/2014**	**26**	**David Trilla,Cristóbal Orte**
33		**Benchmarks**	**11d**	**15/05/2014**	**29/05/2014**		
34		HPL	3d	15/05/2014	19/05/2014		
35		HPCC	3d	19/05/2014	21/05/2014		
36		Aplicaciones ISC	7d	21/05/2014	29/05/2014		
37		**Análisis de rendimiento**	**16d**	**30/05/2014**	**20/06/2014**	**32**	**David Trilla,Cristóbal Orte**
38		Obtención de trazas	3d	30/05/2014	03/06/2014		
39		Análisis con Paraver	7d	03/06/2014	11/06/2014		
40		Análisis mediciones físicas	2d	11/06/2014	12/06/2014		
41		Optimización del rendimiento	7d	12/06/2014	20/06/2014		

Figura 2.1: Listado de Tareas

2.2.2. Diagrama de Gantt

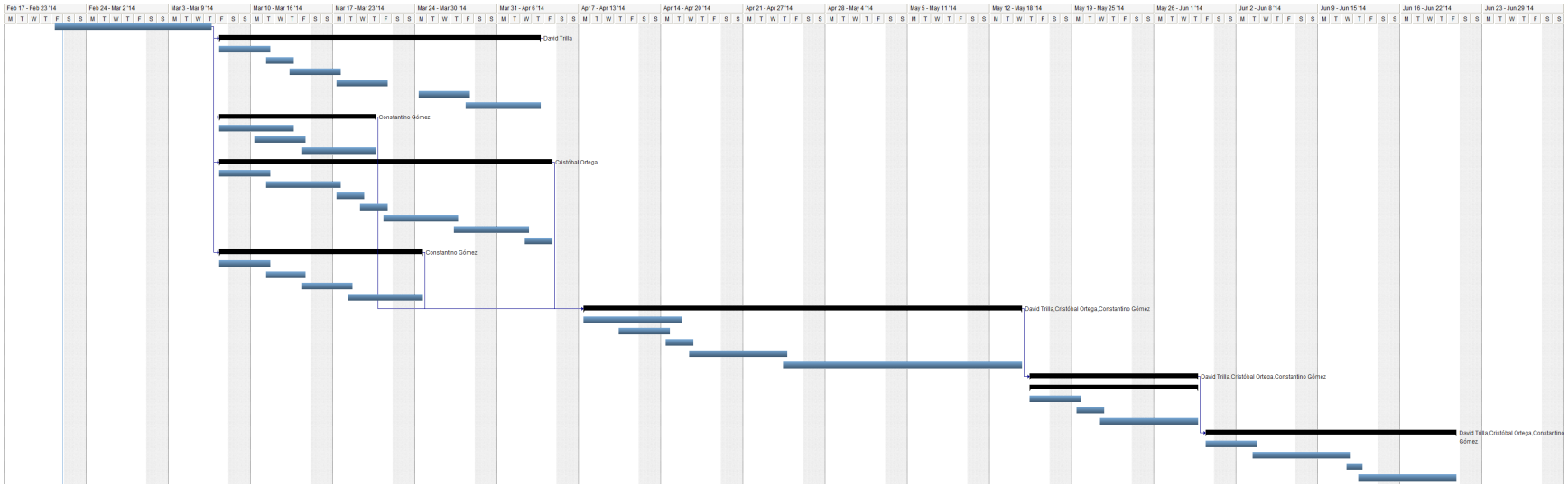

Figura 2.2: Diagrama de Gantt

2.2.3. Recursos

Los recursos que tendremos para este proyecto serán principalmente humanos, tendrán un papel importante para el estudio teórico, el montaje y configuración del clúster.

Para la parte de estudio, nos apoyaremos en publicaciones científicas, revistas y papers, además de sitios online especializados de este tipo de hardware y software. También se hará uso de libros que puedan tratar los temas de HPC o clústeres, pese a que estos se prevén en menor medida.

Para la parte práctica del montaje del clúster dispondremos principalmente del hardware que nos ofrezca el departamento de computadores y el sitio en el que nos permita colocarlo. Para realizar las medidas de consumo energético se ha dispuesto de un medidor de potencia Yokogawa cedido por el Barcelona Supercomputing Center (BSC).

Finalmente, otros recursos utilizados son los ordenadores personales para la redacción de la memoria y conectar en remoto al hardware de desarrollo.

2.3. Presupuesto

El proyecto se basa en montar un clúster y configurarlo de manera correcta para poder ejecutar aplicaciones HPC en él. En este documento se hace una estimación del precio de los recursos que se usarán a lo largo del trabajo. Las amortizaciones se calcularán respecto a 6 meses.

2.3.1. Recursos humanos

Se necesitan técnicos que se encarguen de diseñar, instalar y configurar dicho clúster. Siendo 3 personas en este proyecto, repartiremos las mismas horas para todos. Dentro de cada bloque de horas, cada uno se centrará en una de las divisiones lógicas que hemos establecido: hardware, software de sistema y aplicaciones.

Para las decisiones de elección de componentes se han considerado horas de Director de proyecto. Para el montaje e instalación del clúster se necesitarán competencias de Administrador de Sistema. Para realizar los benchmarks adecuados e interpretar resultados se han considerado horas de Analista. Los datos que utilizamos están basados en portales online de ofertas laborales.

Rol	**Horas estimadas**	**Precio por hora**	**Total por persona**	**Total grupo**
Director de proyecto	125h	50€/h	6250€	18750€
Administrador de sistema	155h	26€/h	4030€	12090€
Analista	95h	30€/h	2850€	8550€
Total	375h			**39390€**

Cuadro 2.2: Costes asociados a recursos humanos.

2.3.2. Hardware

Para la segunda parte del proyecto será esencial el hardware que adquiramos para poder trabajar con él. Datos obtenidos de tiendas online.

Producto	**Precio unitario**	**Unidades**	**Vida útil**	**Amortización total**
Arndale Octa(Exynos 5420)	150€	6	5 años	90€
Fuentes de alimentacion	5€	6	5 años	3€
Tarjetas SD	7.5€	12	5 años	9€
Power meter Yokogawa	1830€	1	15 años	61€
Switch Netgear 100-Mbit	89€	1	10 años	4.45€
1TB Hard Drive Storage	70€	1	7 años	5€
125M cat 5E FTP	68€	1	20 años	1.7€
Total				**174.15€**

Cuadro 2.3: Costes derivados de la compra de Hardware.

Tanto las placas Arndale Octa como las fuentes de alimentación y el medidor de potencia Yokogawa han sido cedidos por el BSC. El resto del material ha sido cedido por el departamento de Arquitectura de Computadores. Al final de la realización del proyecto se espera deshacer el montaje y retornar íntegro todo el material para su posterior reutilización en otras actividades de investigación o proyectos.

2.3.3. Software

El software requerido para la instalación, benchmarking y análisis de la máquina es de acceso gratuito. Gran parte de de los programas tienen licencias de Software Libre, y las que no, disponen de versiones gratuitas con propósito no comercial. Necesitaremos distinto software de sistema para gestionar la máquina como aplicaciones para medir su rendimiento.

2.3.4. Gastos generales

Necesitaremos un sitio donde instalar el clúster físicamente, para ello necesitaremos un sitio con espacio y una instalación eléctrica adecuada además de Internet para tener acceso a los sistemas desde fuera.

El laboratorio C6-103 cedido también por el departamento de AC cuenta con todo lo necesario para la realización del proyecto. Se ha hecho una estimación del coste que supondría disponer de un espacio similar.

Figura 2.3: Estado del Laboratorio C6-103 al llegar.

Concepto	**Coste hora**	**Coste total**
Espacio físico	0.368€	1590€
Electricidad	0.083€	358€
Internet	0.07€	300€
Total		**2248€**

Cuadro 2.4: Desglose de los gastos generales.

2.3.5. Presupuesto total

Concepto	**Coste estimado**	**+ Impuestos**
Recursos humanos	39390€	51600.9€(31 % S.S.)
Hardware	174.15€	210.7€(21 % IVA)
Software	625€	300€(21 % IVA)
Gastos generales	2248€	2720.08€(21 % IVA)
Total	**42437.15€**	**54831.68€**

Cuadro 2.5: Resumen presupuesto total.

Capítulo 3

State of art

3.1. Stack de software en High Perfomance Computing

Si nos fijamos en el TOP500 las 4 empresas que venden más sistemas HPC són: HP, IBM, Cray Inc., SGI.

Estas empresas usan el siguiente stack de software:

3.1.1. HP

En la figura 3.1 vemos el stack que ofrece HP en sus clústers, pero no ofrecen ninguna información realmente útil aparte de que ofrecen Linux y Windows, y tienen software preparado por ellos.

Para ver con mayor detalle el software que ofrecen, miramos un clúster montado por ellos y que esta en la posición 79 del TOP500. El supercomputador se llama Triolith y es usado por el centro nacional de supercomputación de Suecia. (14)

Librerías	Intel math kernel library	
MPI	IntelMPI	OpenMPI
Compiladores	Intel compiler collection	
Sheduler	SLURM	
SO	Linux	

Cuadro 3.1: Software stack de Triolith (14)

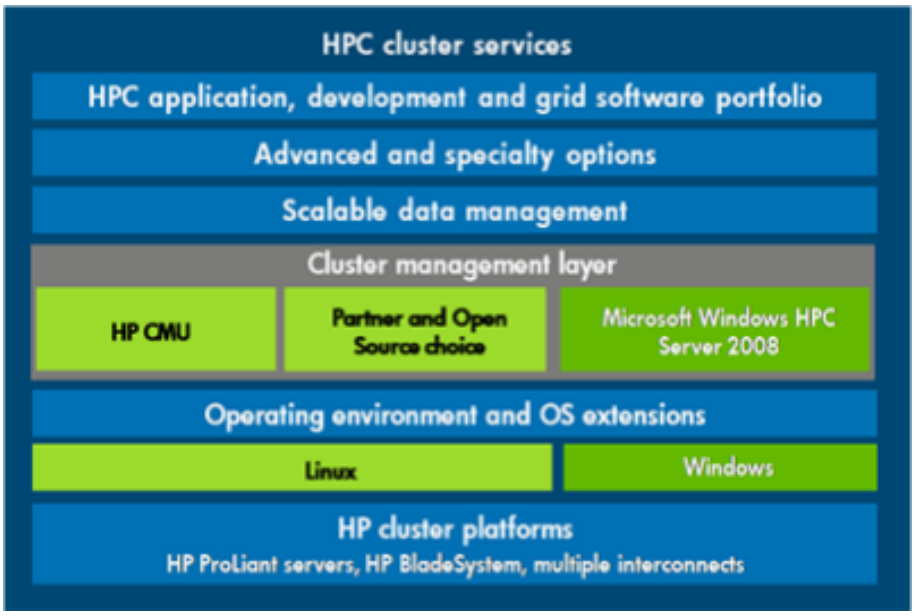

Figura 3.1: Stack de software usado por HP (18)

3.1.2. IBM

Si nos abstraemos de los distintos nodos que IBM usa en sus clústers como vemos en 3.2 y pensamos en un supercomputador donde todos los nodos son iguales tenemos este stack:

Librerías	MASS	FFTW	Netlib
MPI	MPI		
Compiladores	GCC	IBM Compilers	
Sheduler	SLURM		
SO	Linux		

Cuadro 3.2: Software stack de IBM (7)

En los nodos de computación se usa un kernel llamado Compute Node Kernel (CNK), que es el kernel que usa IBM en los Blue Gene, es un kernel muy simple que delega la tarea de entrada/salida a otros nodos, los I/O Nodos, que corren Linux.

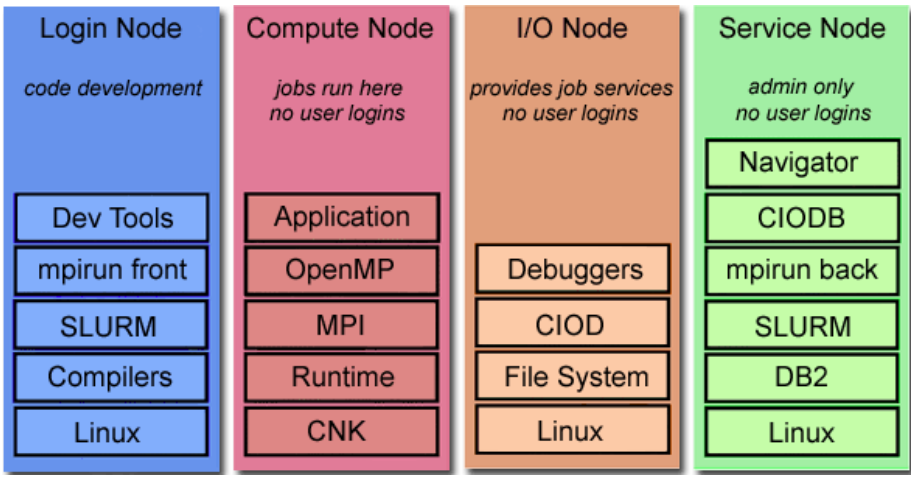

Figura 3.2: Stack de software usado por IBM (7)

3.1.3. Cray Inc.

Lo primero que se destaca de 3.3 es la cantidad de software propio que usan y la cantidad de software que ofrecen.

Performance Monitoring	HPCC	Perfctr	IOR	PAPI/IPM	netperf	
Development Tools	Cray Compiler Environment (CCE)	Intel® Cluster Studio	PGI (PGI CDK)	GNU		
Application Libraries	Cray LibSci	MVAPICH2	OpenMPI	Intel® MPI (Cluster Studio)		
Resource Management/ Job Scheduling	Grid Engine Integrated	SLURM Integrated	MOAB	Altair PBS Professional	IBM Platform LSF	Torque/Maui
File System	NFS	Local FS (ext3, ext4, XFS)	PanFS	Lustre		
Provisioning	Cray® Advanced Cluster Engine (ACE™) Management Software					
Cluster Monitoring	Cray ACE (iSCB and OpenIPMI)					
Remote Power Management	Cray ACE					
Remote Console Management	Cray ACE					
Operating System	Linux (Red Hat, CentOS, SUSE)					

Figura 3.3: Stack de software usado por Cray Inc. (21)

3.1.4. SGI

Del supercomputador Tianhe-2 tenemos la siguiente tabla con el stack de software que usan: (12)

Librerías	Intel math kernel library
MPI	MPICH
Compiladores	Intel compiler collection
Sheduler	SLURM
SO	Linux

Cuadro 3.3: Software stack de Tianhe-2 (12)

Cuentan con librerías optimizadas para las Xeon Phi que usan. También han desarrollado algo llamado OpenMC, que es una capa de abstracción del tipo OpenMP, CUDA o OpenCL para poder facilitar el uso de los procesadores y las Xeon Phi del nodo a la vez

3.2. Sistemas operativos

Actualmente en el TOP500 (35) más del 90 % de sistemas corren una versión de Linux. En cambio, Unix ha ido perdiendo mercado y ahora mismo esta presente sólo en un 2 % de supercomputadores como vemos en la figura 3.4

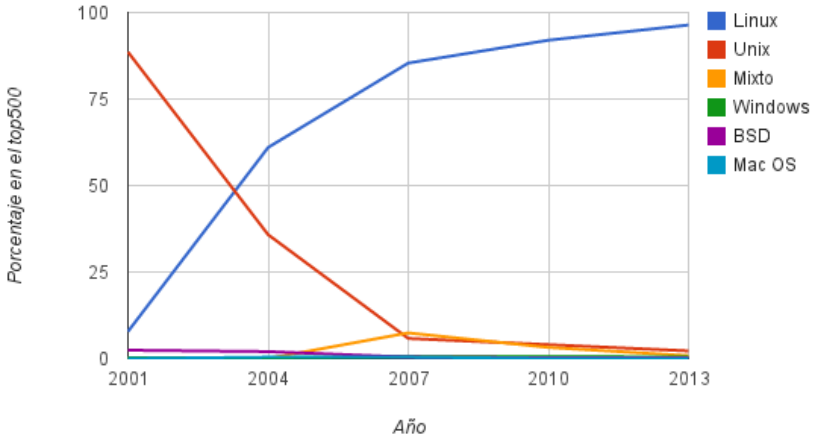

Figura 3.4: Evolución de los SO en el TOP500

Si nos fijamos en la figura 3.5, vemos que Linux está dominando el mercado de supercomputadores, incluso IBM una empresa que está presente en el 32 % de supercomputadores del TOP500 está dejando de lado su SO AIX para adaptarse a Linux, en 2002 la empresa dijo que usarían Linux para sus supercomputadores Blue Gene, ya que el sistema operativo Linux es abierto y podría ser usado en un supercomputador del tamaño del Blue Gene, además de que Linux cuenta con una comunidad de la cuál pueden obtener información. (19) (9)

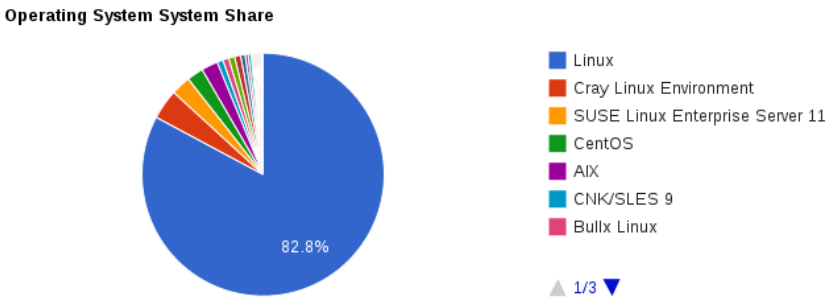

Figura 3.5: Mercado de Sistemas Operativos en el TOP500

Los motivos que hacen a Linux la opción dominante en supercomputadores son (1)

- Modularidad: una de las ventajas de Linux es poder ampliar el kernel con distintos módulos, cosa que hace que sea posible modificar el kernel para conseguir ciertos objetivos como conseguir mayor eficiencia energética o mayor rendimiento.
- Naturaleza del kernel: el kernel de Linux puede ser compilado con una gran variedad de opciones, haciendo que su tamaño se reduzca o tenga más soporte para distintos dispositivos .
- Escalabilidad: cuando tenemos una gran cantidad de nodos necesitamos que el sistema operativo no sea un problema, Linux permite escalar los sistemas, por ello es elegido en muchos servidores y supercomputadores.
- Open source: al ser de código abierto se pueden conseguir kernels personalizados con cosas muy particulares del sistema en el que está, además de arreglar fallos antes de que sean arreglados oficialmente.
- Soporte de arquitecturas: Linux soporta una gran variedad de arquitecturas distintas, esto hace que sea más rápido y fácil de instar y configurar en cualquier sistema.
- Coste: Linux se distribuye de forma gratuita, cosa a tener en cuenta si tenemos 3000 nodos cada uno con un sistema operativo, que en el caso de que hubiese que pagar una licencia por cada instancia del sistema operativo la inversión inicial sería mayor.

3.3. Monitorización

Cuando tratamos con el tipo de hardware que estamos tratando queremos saber porqué estamos obteniendo los resultados que estamos consiguiendo, y así poder identificar el problema y poder solucionarlo. Cuando se trata de una sola máquina de un sólo nodo o pocas nodos, que corren Linux por ejemplo, tenemos herramientas como top, vmstat, free, iostat, df, netstat y muchas más.

Pero cuando estamos tratando con una máquina que tiene cientos de nodos no podemos conectarnos una por una para ejecutar los comandos y después parsear la información ya que sería una gran cantidad de información. Por otro lado podemos usar un software de monitorización para automatizar este proceso.

Cuando hablamos de software de monitorización podemos distinguir 2 grupos:

3.3.1. Monitorización de red

Este tipo de software se encarga de ver el estado de servicios, aplicaciones, servidores... este tipo de software es útil si queremos saber en todo momento el estado de nuestra red, si los servidores están corriendo o si los distintos servicios que tengamos están up.

En este apartado los software más usados son:

- **Nagios** (13), ampliamente usado es usado para conocer el estado desde el uso de recursos de una máquina hasta servicios de red. Dispone de un sistema de plugins con el que poder personalizar el sistema y conocer al detalle cualquier dato deseado. Cuenta con un sistema de alertas, al sobrepasar algún límite de uso de CPU o si un servicio está caído puede ser programado para tomar alguna acción y avisar a los responsables. Bajo licencia GNU General Public License, aunque venden conjunto de software ya preparado con distintos añadidos.

- **MRTG** (30), siglas de Multi Router Traffic Grapher, es un software más orientado al análisis de las interfaces de red de distintos dispositivos, switchs, routers... dando información sobre entrada y salida en bytes de la interfaz del dispositivo. También cuenta con un servició de alertas para la notificación de posibles problemas. Bajo licencia GNU General Public License.

- **Zabbix** (33), como Nagios, combina la monitorización de red con la de sistema de los hosts añadidos al sistema, también dispone de un sistema de alertas para poder informar si un servicio cae o un host usa mucha CPU. Bajo licencia GNU General Public License.

Por supuesto, estos 3 programas son usados, pero hay un gran abanico de software que realiza si no bien los mismo, algo muy parecido, sólo cambiando como recogen los datos, como se procesan o el front-end final.

3.3.2. Monitorización de sistema

Este tipo de software trabaja más a nivel de máquina, mostrándonos información sobre la propia máquina, como por ejemplo el uso de cpu, de memoria o el uso de la red. Útil si queremos ver que sucede en una máquina al ejecutar una aplicación y saber porqué no obtenemos un mejor resultado.

- **Ganglia** (31), es un monitor de sistema orientado a HPC, es escalable y distribuido, mide todo el uso de CPU, memoria, red...además se le pueden añadir más métricas para que sean recogidas. Es usado por muchos supercomputadores y es escalable hasta 2000 nodos. Usa el concepto de que los host recogen los datos y son enviados al servidor donde se recogen y son procesados para ser mostrados en el front-end. Bajo licencia BSD.

- **Supermon** (34), muy parecido a Ganglia, orientado también a HPC, y recoge métricas de recursos locales. Supermon también hace que los hosts sean los encargados de recoger los datos locales y enviarlos a un servidor (los hosts son llamados mon y el servidor supermon). La diferencia con Ganglia en este aspecto es que Supermon no realiza el paso de información con multicast y el servidor necesita saber de antemano cuántos y qué nodos les pasará información, es decir, hay que registrar cada nuevo nodo. Ganglia al contrario permite multicast y no necesita saber cuantos nodos hay o habrá en la red, simplemente recoge todos los mensajes que sean enviados al servidor (11)

- **Cacti** (16), puede ser usado como monitor de red como de sistema, conociendo el consumo de recursos de los distintos servidores que tengamos hasta el estado de servicios de red ya que también dispone de un sistema de plugins. Usa un sistema donde el servidor Cacti hace polling sobre los nodos a consultar.

3.4. Software de desarrollo

En cualquier sistema donde haya que compilar, debuggar y probar cosas necesitamos de unas herramientas que nos faciliten esta tarea, teniendo en cuenta que estamos tratando con un clúster y más aún, un clúster que podríamos decir que es de desarrollo, donde se compilarán cosas, se eliminarán cosas, se probarán cosas...

A nivel de programa tenemos herramientas como GNU Debugger (GDB) que nos permiten debuggar, pero en una máquina como es un clúster con el software que se ejecutará en él como es el software de la competición SCC que ya sabemos que funciona y en principio está optimizado, no debemos fijarnos tanto a nivel de programa sino más a nivel de comunicación entre nodos, Message Passing Interface (MPI), uso de recursos y demás.

Por ello necesitamos diferenciar 2 grandes grupos, uno de compiladores, pues los necesitaremos para poder compilar el distinto software con optimizaciones dependientes de la arquitectura y software de profiling, para poder ver porqué cierta ejecución está dando unos malos resultados.

Todo fabricante de hardware tiene su propia suite de compiladores y debuggers para facilitar la creación de aplicaciones para sus sistemas.

3.4.1. Compiladores

El más conocido en el mundo Linux es quizá GNU Compiler Collection que ofrece una gran cantidad de flags para compilar según la micro arquitectura del procesador, o el Instruction Set Architecture o en castellano conjunto de instrucciones (ISA) de éste, o si queremos usar instrucciones Single Instruction, Multiple Data o en castellano una instrucción, múltiples datos (SIMD) por ejemplo. Mantenido por el Proyecto GNU, bajo licencia GPL.

Tiene soporte para diversas arquitecturas, entre las cuales están ARM, MIPS, x86, x86-64 entre otras.

Si después miramos por fabricante, por los 2 grandes a nivel de arquitectura x86, tenemos:

Intel, con su compilador Intel compiler collection que según su página web es más rápido que sus competidores directos (23)

Figura 3.6: Perfomance de Intel compiler collection (23)

El problema de este conjunto de compiladores de Intel es que si nuestro procesador no en fabricado por Intel y compilamos un programa para x86 el binario resultante será distinto a si compilamos sobre un procesador fabricado por Intel. En otras palabras, Intel compiler collection genera código no óptimo si estamos sobre procesadores no fabricados por Intel (22).

AMD, recomienda Open64 Compiler Suite, el desarrollo de esta suite es entre distintas empresas como AMD, Nvidia entre otras, bajo licencia GPL. AMD realiza un fork del proyecto para optimizarlo para la arquitectura x86, independientemente si es sobre un procesador Intel o AMD. Nvidia también realiza otra fork para optimizar la programación en Compute Unified Device Architecture. (4)

En otro tipo de arquitecturas tendríamos a:

IBM, con su suite XL C and C++ compilers que está para arquitecturas de x86 como para su arquitectura propia Power, para los supercomputadores Blue Gene y para sistemas operativos como Linux o su propio AIX. Su uso es bajo pago de la licencia. (20)

ARM, cuenta con ARM Compiler, aunque dentro de su IDE ARM DS-5 permiten cambiar el compilador por gcc, ya que ARM Compiler es de versión de pago (al igual que el IDE). (6)

3.4.2. Profiling

En cuanto a crear trazas de un programa MPI el software que más conocemos es desarrollado por el BSC como Extrae y Paraver.

Extrae permite crear trazas de programas MPI a través de instrumentalizar el código, que luego puedan ser analizadas por Paraver. Es compatible con la mayoría de versiones de MPI que hay actualmente.

3.5. Software de scheduling

Cuando tenemos un sistema que potencialmente será usado por muchos usuarios luchando por los distintos recursos hardware que tenemos si queremos ofrecer un mejor servicio o rendimiento a esos usuarios, tenemos que limitar los recursos.

Para ello se suelen instalar sistemas de colas para repartir los recursos hardware, entonces los usuarios mandan un pequeño script con los distintos recursos que necesitan y lo que quieren ejecutar al sistema de colas, el sistema de colas controla cuando tal cantidad de recursos está disponible y ejecuta lo que el usuario realmente quería.

Algunos gestores de colas conocidos son:

Torque, basada en otro gestor de colas antiguo llamado OpenPBS, Torque es open-source desarollado por Adaptive Computing. Su sistema de colas es FIFO (first in, first out), aunque es posible cambiar como lo gestiona. (2)

Un ejemplo de un script para enviar a ejecutar algo al sistema sería este:

```
#!/bin/bash
#PBS -l nodes=2:ppn=2

cd $HOME/programa_mpi
mpirun -np 4 ./programa_mpi
```

Donde la opción -l es para indicarle cuántos nodos y cuántos procesadores por nodo usaremos. Luego simplemente iría el comando que queremos ejecutar. La salida estándar y de error por defecto se guardan en el home del usuario que llama a Torque.

SLURM, también es open-source, desarrollado por Lawrence Livermore National Laboratory, diseñado para poder correr en grandes clústers (según ellos hasta decenas de millones de procesadores). Más sencillo de configurar que Torque, al igual que éste es posible cambiar su política de cola por otra, por defecto usa también FIFO.(28)

```
#!/bin/bash
#SBATCH -N 2
#SBATCH -n 2

cd $HOME/programa_mpi
mpirun -np 4 ./programa_mpi
```

La opción N nos indica el número de procesadores a usar, y n el número de threads por procesador a usar.

3.6. Message Passing Interface

MPI es un estándar para el paso de mensajes explícitos, en este estándar se definen la sintaxis de las funciones que una librería debe contener para cumplir con el estándar. Al ser mensajes explícitos, en el código del programa debemos tener en cuenta de cuántos procesadores disponemos para poder realizar el paso de mensajes entre ellos.

MPI usa el concepto de procesos, por tanto si ejecutamos una aplicación con 10 procesos puede ser que se pongan en el mismo nodo, para ello hay que definir cuántos procesos puede tomar un propio nodo. En opemMPI (32) se definiría así si contáramos con un nodo que dispone de 4 cores:

```
user@NODO1 slots=4 max_slots=4
user@NODO2 slots=4 max_slots=4
```

En el siguiente código vemos un ejemplo de programa en MPI (38)

```
/*
  "Hello World" MPI Test Program
 */
 #include <mpi.h>
 #include <stdio.h>
 #include <string.h>

 #define BUFSIZE 128
 #define TAG 0

 int main(int argc, char *argv[])
 {
   char idstr[32];
   char buff[BUFSIZE];
   int numprocs;
   int myid;
   int i;
   MPI_Status stat;
   MPI_Init(&argc,&argv);
```

```
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);


    if(myid == 0)
    {
      printf("%d: We have %d processors\n", myid, numprocs);
      for(i=1;i<numprocs;i++)
      {
        sprintf(buff, "Hello %d! ", i);
        MPI_Send(buff, BUFSIZE, MPI_CHAR, i, TAG, MPI_COMM_WORLD);
      }
      for(i=1;i<numprocs;i++)
      {
        MPI_Recv(buff, BUFSIZE, MPI_CHAR, i, TAG, MPI_COMM_WORLD, &
            stat);
        printf("%d: %s\n", myid, buff);
      }
    }
    else
    {
      /* receive from rank 0: */
      MPI_Recv(buff, BUFSIZE, MPI_CHAR, 0, TAG, MPI_COMM_WORLD, &stat
          );
      sprintf(idstr, "Processor %d ", myid);
      strncat(buff, idstr, BUFSIZE-1);
      strncat(buff, "reporting for duty\n", BUFSIZE-1);

      /* send to rank 0: */
      MPI_Send(buff, BUFSIZE, MPI_CHAR, 0, TAG, MPI_COMM_WORLD);
    }


    MPI_Finalize();
    return 0;
}
```

Este código hace lo siguiente:

- Cada proceso pregunta al runtime de MPI cuántos procesos hay en ejecución y que identificador tiene, identificado como myid.
- Si es el proceso identificado con el número 0 entonces envía un “Hello” al respectivo proceso
- Los demás procesos con identificador distinto a 0 esperan el “Hello” del proceso 0
- Una vez recibido el “Hello” concatenan una frase y vuelven a enviársela al proceso 0
- El proceso 0 recoge todos estos mensajes y los escribe por pantalla

Por tanto, la salida del programa con 4 procesos sería:

```
0: We have 4 processors
0: Hello 1! Processor 1 reporting for duty
0: Hello 2! Processor 2 reporting for duty
0: Hello 3! Processor 3 reporting for duty
```

3.6.1. Implementaciones MPI

Hay 2 grandes librerías que implementan MPI:

MPICH

Una versión que es ampliamente usada por distintas empresas/universidades para crear su versión de MPI, como por ejemplo IBM Platform MPI, Intel MPI o MVAPICH entre otros.

Actualmente está presente en 9 de los 10 supercomputadores más rápidos del mundo (35) (25) (26)

Tiene 2 objetivos:

- Ofrecer una implementación MPI eficiente que soporte arquitecturas
- Permitir investigación en MPI con módulos fácilmente ampliables.

OpenMPI

Este proyecto de MPI es la combinación de distintos proyectos como FT-MPI, LA-MPI, LAM/MPI y PACX-MPI. (32)

Fue creado con los siguientes objetivos:

- Crear una implementación de MPI gratis y libre.
- Ofrecer un gran rendimiento
- Involucrar la comunidad de HPC
- Evitar el problema de crear derivados (forks), común en otros proyectos de MPI
- Dar soporte un amplio abanico de plataformas HPC

3.7. Librerías

En cuanto a librerías matemáticas que necesitamos, dependerá de las aplicaciones que vayamos a ejecutar en el clúster.

Como nos vamos a presentar al SCC necesitamos las librerías para ejecutar el LINPACK y así poder sacar mejores resultados, entonces las librerías que necesitamos son ATLAS y FFTW, que son las más conocidas.

Aunque cada fabricante suele optimizar estás librerías para su propia arquitectura, creando conjuntos de librerías matemáticas, como Intel con su Intel math kernel library, que dicen tener la librería matemática más rápida para procesadores Intel y compatibles (24), o AMD con su AMD Core Math Library(3).

3.7.1. ATLAS

Según la web del proyecto: "proporciona interfaces para C y Fortran para una implementación portable y eficiente de BLAS, como también algunas rutinas de LAPACK"(37)

Este software nos permite generar librerías de BLAS y LAPACK optimizadas para la máquina en la que queremos correr tales librerías, aparte de tener unos flags de compilación genéricos por arquitectura, nos permite ajustarlos a nuestro parecer para poder optimizar más allá de flags genéricos. Bajo licencia BSD.

Intel por ejemplo ha realizado una comparación entre ATLAS y su Intel math kernel library(5):

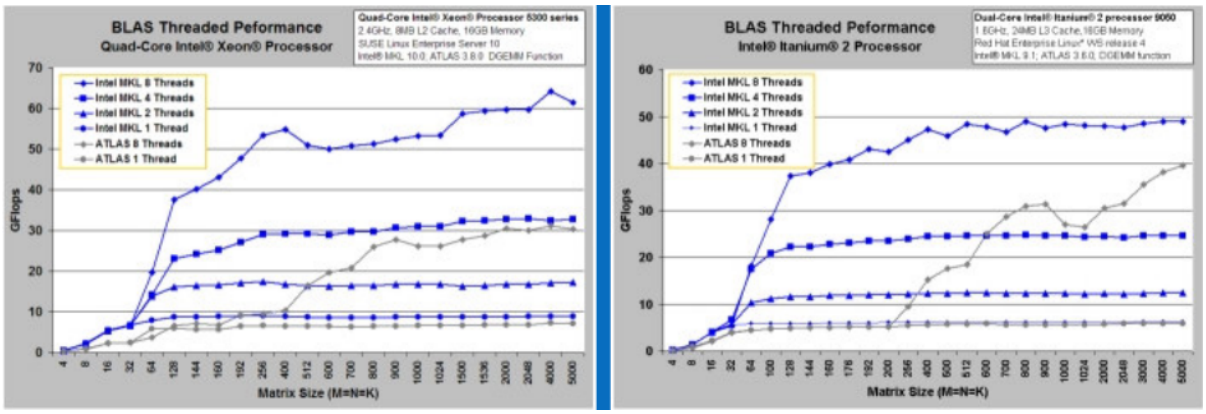

Figura 3.7: Comparación entre ATLAS y MKL (5)

Vemos que Intel saca un rendimiento bastante mayor al ATLAS, pero veamos que pasa si también se ejecuta en un AMD:

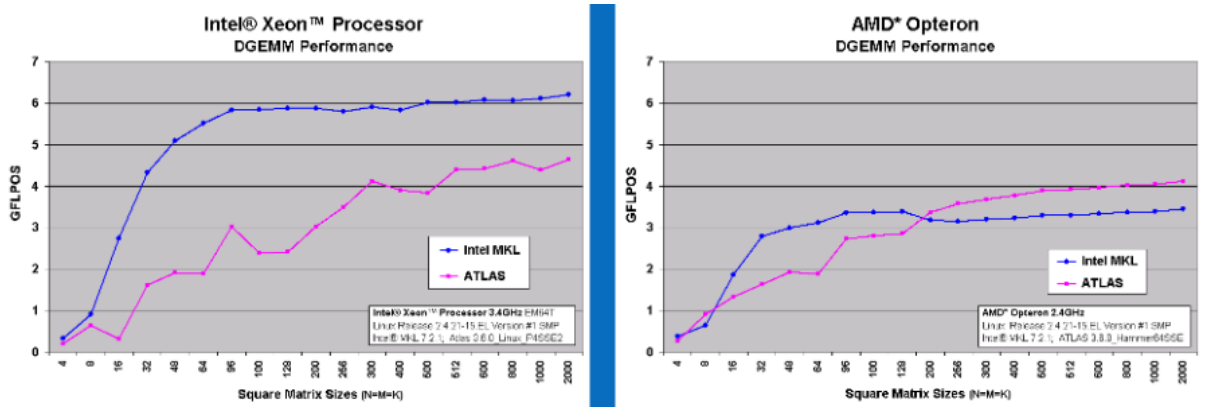

Figura 3.8: Comparación entre ATLAS y MKL (5)

Aquí el ganador en rendimiento no está tan claro ya que en la plataforma de AMD MKL gana sólo para matrices pequeñas, pero a partir de 200 elementos ATLAS sigue escalando mientras MKL mantiene el mismo rendimiento. Por lo que sería aconsejable que si se duda entre un software u otro correr algunas pruebas si es posible.

3.7.2. FFTW

Librería para calcular la transformada rápida de Fourier, es conocida por ser la más rápida en cuanto a software gratuito ya que esta bajo licencia GPL (15).

Muchos fabricantes también tienen su versión optimizada de FFTW, Intel también compara su Intel math kernel library con FFTW:

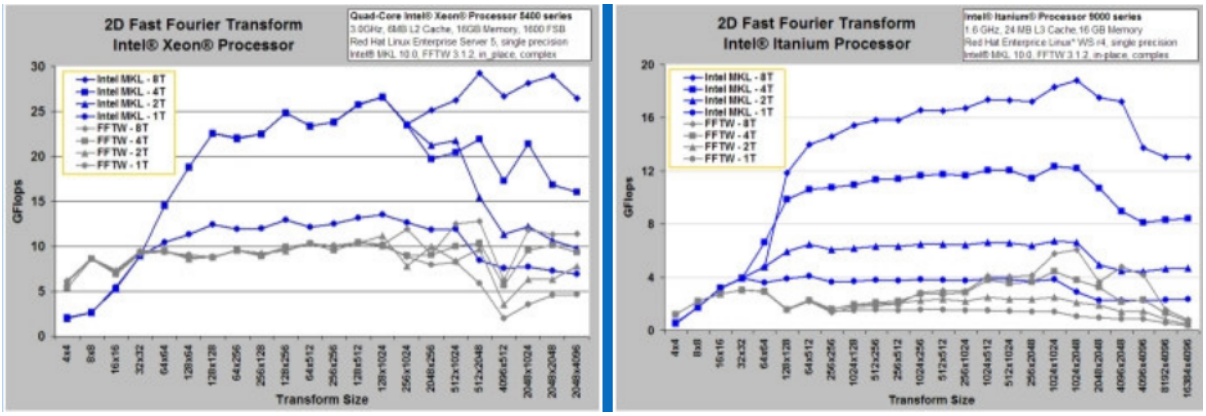

Figura 3.9: Comparación entre FFTW y MKL (5)

Aquí la librería FFTW se queda atrás en cuanto a rendimiento, pero sólo hay pruebas con procesadores Intel, podría ocurrir, que si probamos contra procesadores x86 que no sean de Intel es posible que se vea un comportamiento como el de la figura 3.8

Capítulo 4

Diseño de un clúster

4.1. Clúster a usar

El hardware que tenemos montado para usar como clúster es el siguiente: (36)

- Nodo: Arndale Octa (x6)
  - System On Chip (SoC): Samsung Eynos 5420
    - Central Processing Unit (CPU): ARM Cortex-A15 (x4) + ARM Cortex-A7 (x4)
    - Graphics Processing Unit (GPU): Mali T-628
    - Memoria: 2 Gigabyte (GB) 933 Megahercios (MHz)
  - 100 Mega Bits Por Segundo (Mbps) Interfaz Ethernet
  - Almacenamiento:
    - 4 GB Embedded Multi-Media Card (eMMC)
    - 16 GB Secure Digital (SD)
- Switch: NetGear FS524
- Cables Ethernet Cobre (x6)
- Transformadores (x6)

Algunas imágenes del resultado del pequeño montaje:

Figura 4.1: Vista frontal del clúster (cerveza para comparar la escala)

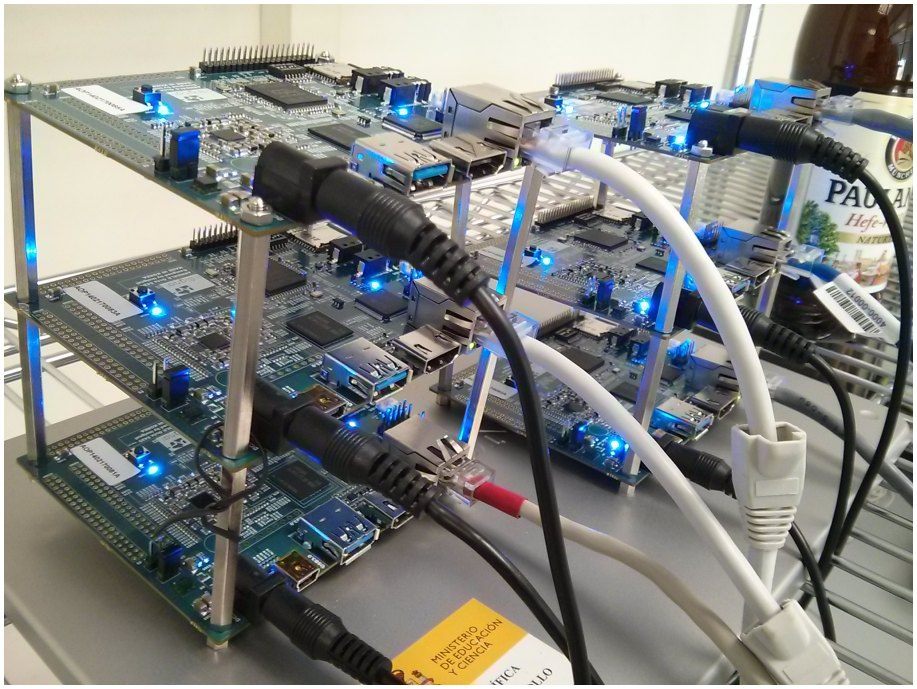

Figura 4.2: Vista lateral del clúster (cerveza para comparar la escala)

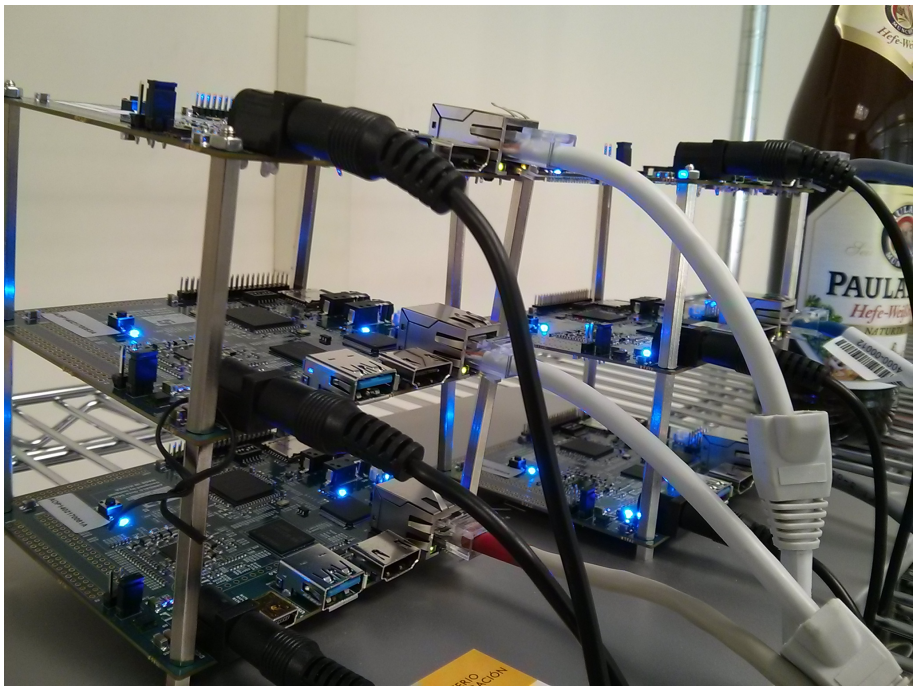

Figura 4.3: Vista cercana del clúster

Figura 4.4: Vista del *switch* de interconexión

4.2. Selección de software

Una vez tenemos el hardware y la red, tenemos que saber que software de sistema instalar, este será nuestro stack de software:

Librerías	ATLAS	FFTW
MPI	OpenMPI	MPICH
Software de desarrollo	GCC	Extrae
Monitorización	Ganglia	
Sistema operativo	Ubuntu	

Cuadro 4.1: Stack de software a usar

Lo ideal en un clúster final sería incluir un gestor de colas y un sistema de alertas monitorizando los nodos. Se ha decidido no instalar ambas cosas por el hecho que estaremos probando distintas cosas en los nodos y un sistema de colas entorpecería las ejecuciones. Y un sistema de monitorización que sea capaz de notificarnos si se nos cae un nodo no es necesario pues estaremos trabajando con ellos constantemente nosotros mismos.

4.3. Sistema operativo

Para el sistema operativo usaremos Linaro. Linaro es un proyecto que pretende unificar esfuerzos para portar proyectos de software libre a ARM (a SoC específicos como pueden ser nuestras placas Arndale Octa), lo que nos interesa de este proyecto son los kernel Linux que cada cierto tiempo sacan añadiendo características que incorporan los chips ARM pero no están integrados en el kernel de Linux (27).

Además de los kernels, nos interesan herramientas que proporcionan para instalarlos en dispositivos, como por ejemplo las SD que usan nuestros nodos para bootar.

Incluso en la web de Linaro, aparte de las versiones soportadas oficialmente del kernel, hay versiones de developers y comunidad, en estas versiones encontramos Ubuntu, que incluye todo lo que sería un Ubuntu normal pero compilado para los distintos SoC que soportan los kernel de Linaro.

Por tanto, por comodidad y eficiencia elegimos usar una versión de **servidor** de Ubuntu, esto nos proporciona ventajas como que contamos detrás con una comunidad grande como es la de Ubuntu, contamos con sus repositorios, y lo mayor ventaja, contamos con algo ya creado, si no tuviéramos esta facilidad, tendríamos que construir nuestra propia distribución a partir del kernel, cosa que nos llevaría bastante más tiempo.

Pero esta solución también tiene desventajas, empezamos con un sistema *sucio* ya que habrá paquetes de software que no necesitamos y que en cambio podrán potencialmente comer recursos. Además, el kernel se habrá compilado con ciertos flags, que quizá no sean óptimos para nosotros.

La última imagen estable que hemos generado está basada en Ubuntu 14.01 versión Servidor, para generarla tenemos 2 opciones que nos ofrece Linaro,

- 1. La imagen .iso preparada para grabar a la sd mediante un simple dd.
- 2. Si la versión ha sido construida hace poco nos dan el binario para que podamos crear la imagen nosotros.

Para la segunda opción necesitamos herramientas de Linaro para crearla:

Como estamos trabajando sobre una versión de Ubuntu Desktop, podemos obtener tales herramientas de la siguiente manera:

```
# add-apt-repository ppa:linaro-maintainers/tools
# apt-get update
# apt-get install linaro-image-tools
```

Después necesitamos el conjunto de paquetes auxiliares dependientes de nuestro SoC para poder crear la imagen:

```
$ wget http://releases.linaro.org/14.01/ubuntu/arndale-octa/
    hwpack_linaro-arndale-octa_20140126-596_armhf_supported.tar.gz
```

Y por último necesitamos el binario de la versión de Ubuntu que queremos:

```
$ wget http://releases.linaro.org/14.01/ubuntu/arndale-octa/linaro-
    saucy-server-20140126-627.tar.gz
```

Si inspeccionamos *http://releases.linaro.org/14.01/ubuntu* podremos ver en que otros SoC podemos tener Ubuntu 14.01.

Seguidamente y con la tarjeta SD insertada e identificada por su device (hacer *dmesg* al insertarla) creamos la imagen:

```
# linaro-media-create --rootfs ext3 --mmc /dev/$SD --binary linaro-
    saucy-server-20140126-627.tar.gz --hwpack hwpack_linaro-arndale-
    octa_20140126-596_armhf_supported.tar.gz --dev arndale-octa
```

Donde *$SD* es igual al device ocupado por la SD (sdX en /dev/)

Una vez este último comanda ha acabado, ya podemos insertar la SD y encender el nodo.

Para el primer boteo es necesario hacerlo mediante cable serie, una vez uboot (pequeña BIOS que tienen los nodos) ha inicializado todo y nos deja mandar comandos, necesitamos hacer boot mediante: *fastboot*

Una vez inicializado el sistema operativo tendremos como único usuario linaro/linaro, que está en la lista de *sudoers*. Por tanto, creamos un nuevo usuario que será el que usaremos:

```
//Creamos el usuario
# adduser administrador
# usermod -a -G sudo administrador
```

Lo anterior necesitaremos estar por serie, puesto que no tenemos el ssh-server instalado, para arreglarlo, instalamos los siguientes paquetes con sus dependencias:

- openssh-server, para poder conectarnos a los nodos por ssh.
- ntpd, para que todos los nodos tengan la misma hora, ya que no se guarda la fecha en las placas después de apagarlas.

- binutils,gcc y make, ya que los necesitaremos más tarde.

```
# apt-get install openssh-server ntpd binutils gcc make
```

Instalamos *VIM* y sus dependencias:

```
# apt-get install libgpm2 libpython2.7 vim vim-runtime
```

Borramos servicios que no necesitamos y que seguramente consuman recursos de CPU, red y memoria:

```
# apt-get purge apache2 apache2-mpm-prefork php5-mysql libapache2-
    mod-php5 mysql-server
# apt-get autoremove
```

Para facilitar la administración de nodos en remoto, desactivamos que el comando *sudo* nos pida contraseña:

```
# visudo -f /etc/sudoers
%sudo ALL=(ALL:ALL) ALL por %sudo ALL=(ALL:ALL) NOPASSWD:ALL
```

Esto hace que los usuarios que pertenecen al grupo *sudo* puedan usar *sudo* sin contraseña.

Instalamos el cliente de Network File System (NFS) y montamos la unidad compartida por NFS que sabemos que está en el servidor con dirección *192.168.0.1* en los nodos para poder compartir ficheros:

```
$ mkdir /media/ancient
# apt-get install libevent-2.0-5 libgssglue1 libnfsidmap2 libtirpc1
    nfs-common rpcbind
# echo 192.168.0.1:/ancient    /media/ancient   nfs      rsize=8192,
    wsize=8192,timeo=14,intr >> /etc/fstab
# mount -a
```

Durante las pruebas en el clúster nos dimos cuenta que el proceso que comprueba si hay actualizaciones durante cierto tiempo consumía toda la CPU, y ese tiempo no era menospreciable, por tanto, desactivamos la comprobación de actualizaciones:

```
# vim /etc/update-manager/release-upgrades
//Y cambiamos la linea de Prompt=normal a Prompt=never
```

Para cambiar fácilmente el *hostname* de la máquina, ya que trabajamos con una imagen y grabamos en consecuencia el resto, por lo que el *hostname* será el mismo si no se cambia, y puede ser lioso, tenemos disponible un script (en apéndice A) en ancient/scripts:

```
# /media/ancient/scripts/change_hostname
```

Una vez ejecutado este script, el nodo se reiniciará, y para más tarde poder usar MPI ejecutamos el siguiente script (en apéndice B):

```
# /media/ancient/scripts/copia-ids-ssh-host
```

Y ya tendremos un sistema limpio y listo para instalar el próximo software.

4.4. Monitorización

Para monitorizar nuestro clúster optamos por la solución que parece bastante extendida: Ganglia.

La instalación se ha realizado a través de repositorios ya que al no ser pieza clave al medir el rendimiento no nos hace tener que preocuparnos de que estén totalmente optimizado para nuestra arquitectura.

4.4.1. Ganglia

Tenemos que tener en cuenta que los nodos harán de *esclavos* y el servidor que está en *192.168.0.1* hará de *Master*, éste último será el que se encargará de procesar los datos y mostrarlos vía web.

Master En nuestro clúster hará de Master el pc de sobremesa que tenemos como servidor de NFS y DHCP en 192.168.0.1, para así quitarle trabajo a los nodos y ver todo su potencial real. Primero necesitamos instalar dependencias, esto incluye, servidor web (apache), y php para el webfront de Ganglia:

```
# apt-get install libconfuse-common libconfuse0 libdbi1 libganglia1
  librrd4 rrdtool apache2 php5 libapache
```

Después instalamos los 3 paquetes que componen Ganglia:

- ganglia-monitor: es el daemon que se encarga de recoger la información de estado del nodo y enviarla al master.
- ganglia-webfrontend: es el encargado de procesar información y generar la interfaz web
- gmetad: corre en el master y es el que tiene la tarea de ir recogiendo datos de los nodos

```
# apt-get ganglia-monitor ganglia-webfrontend gmetad
```

Y configuramos apache para que Ganglia sea accesible:

```
# cp /etc/ganglia-webfrontend/apache.conf /etc/apache2/sites-enabled
  /ganglia.conf
```

Ahora tenemos que configurar el daemon gmetad para indicarle cada cuánto consultar a los nodos y en qué puerto:

```
# vim /etc/ganglia/gmetad.conf
Cambiar la linea: data_source ``mycluster" 50 192.168.0.1:8649
```

Donde *mycluster* es el nombre de clúster que aparecerá en la web, 50 es cada cuánto recogeremos datos, y después la IP del master con el puerto por el cuál escuchar.

Ahora editamos el ganglia-monitor del master: Ahora tenemos que configurar el daemon gmetad para indicarle cada cuánto consultar a los nodos y en qué puerto:

```
# vim /etc/ganglia/gmond.conf
```

Y cambiamos la definición de nuestro clúster y que puertos usaremos:

```
globals {
  daemonize = yes
  setuid = yes
  user = ganglia
  debug_level = 1
  max_udp_msg_len = 1472
  mute = no
  deaf = no
  host_dmax = 300 /*secs */
  cleanup_threshold = 300 /*secs */
  gexec = no
  send_metadata_interval = 0
}

cluster {
  name = ``VOLVO''
  owner = ``Gaben''
}

udp_send_channel {
  host = 192.168.0.1
  port = 8649
  ttl = 1
}

udp_recv_channel {
  port = 8649
}

tcp_accept_channel {
  port = 8649
}
```

Y ya tenemos el Master configurado, nos falta reiniciar los servicios para que carguen la nueva configuración:

```
# /etc/init.d/ganglia-monitor restart
# /etc/init.d/gmetad restart
# /etc/init.d/apache2 restart
```

Nodos La configuración es muy parecido al Master, pero con menos servicios.
Dependencias:

```
# apt-get install libconfuse-common libconfuse0 libganglia1
```

Sólo nos hará falta instalar el ganglia-monitor:

```
# apt-get install ganglia-monitor
```

Y cambiar la configuración en el fichero /etc/ganglia/gmond.conf

```
globals {
daemonize = yes
setuid = yes
user = ganglia
debug_level = 0
max_udp_msg_len = 1472
mute = no
deaf = yes
host_dmax = 0 /*secs */
cleanup_threshold = 300 /*secs */
 gexec = no
send_metadata_interval = 30
}

cluster {
name = ``VOLVO"
owner = ``Gaben"
}

udp_send_channel {
mcast_join = 192.168.0.1  -> a quien enviarle los datos
port = 8649
ttl = 1
}
```

Importante:

- Cambiar el valor de **deaf** de no a yes, ya que sino leerá los distintos paquetes que son enviados por otros nodos, y en las pruebas llegaban a saturar la CPU.
- Comentar o eliminar las secciones de **udp_recv_channel** ya que no queremos recibir de nadie.

Y si vamos a la dirección http://192.168.0.1/ganglia, veremos los resultados: 4.5

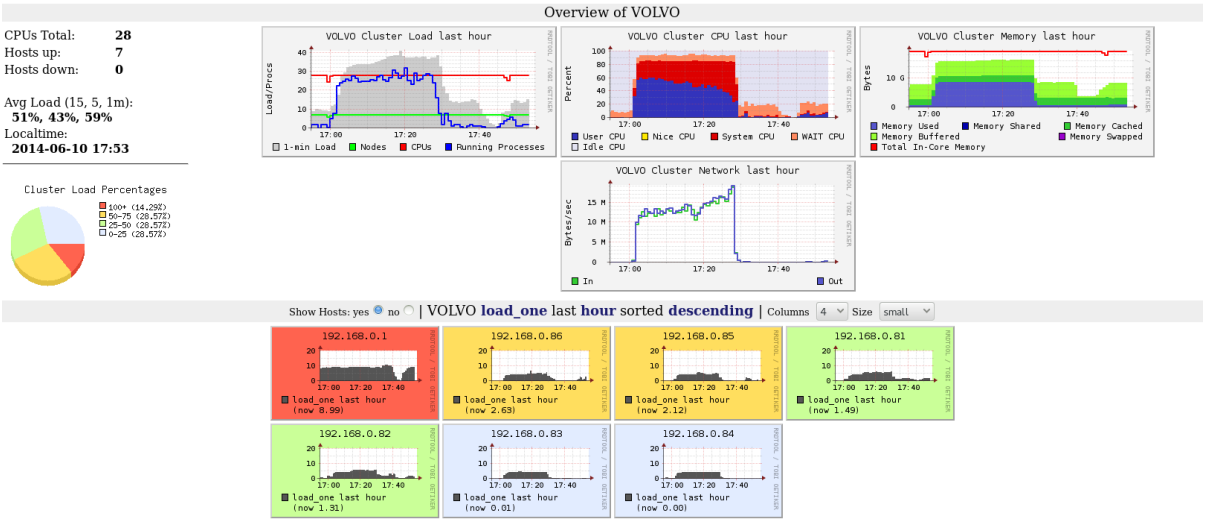

Figura 4.5: Web principal de Ganglia

También podemos ver información a nivel de nodo: 4.6

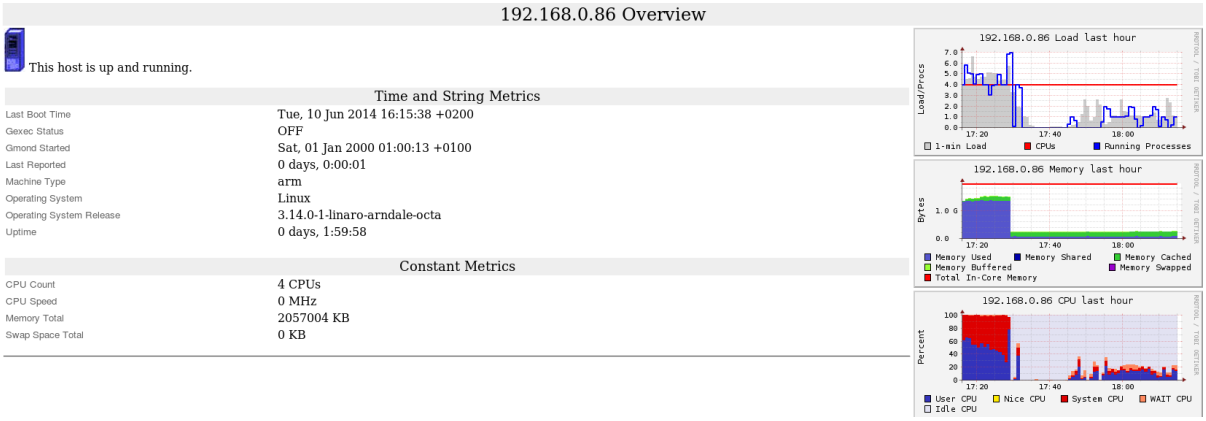

Figura 4.6: Información del nodo 86

4.5. Software de desarrollo

Como software de desarrollo hemos escogido los programas en los cuáles nos sentimos más cómodos, como son los compiladores de GNU, gcc, g++,gfortran... y como herramienta de profiling de software paralelo Extrae, puesto que con anterioridad hemos trabajado con él.

4.5.1. GCC

Para instalar GCC hemos optado por bajarlo por repositorios de Ubuntu, ya que no es una parte vital del rendimiento de las aplicaciones, lo vital es el código que compile, para ello buscaremos los flags que mejor se adapten a nuestra arquitectura.

Para instalar gcc sólo basta con:

```
#apt-get install binutils gcc g++ gfortran
```

Y los flags que usaremos ya sea para C, C+ o fortran:

- **-O3** Nivel de optimización del código que hará.
- **-mcpu=cortex-a15** Indica que el nombre del procesador ARM que se usará, gcc lo utilizará para saber qué tipo de instrucciones puede generar.
- **-mtune=cortex-a15** Le indicamos que puede optimizar el código para este procesador. La diferencia con mcpu es que ésta se encarga de que instrucciones generará, y mtune a que se adaptará gcc para optimizar, aun con instrucciones generados por mcpu.
- **-mfloat-abi=hard** Decimos que tipo de Application binary interface (ABI) usar, usaremos hard, que implica que todo se haga con hardware de coma flotante.
- **-mfpu=neon** Especificamos que hardware de coma flotante tenemos disponible.

4.5.2. Extrae

Instalar Extrae mediante repositorio no es posible, así que tenemos satisfacer las dependecias que tenga, mediante repositorios o código fuente, que nos indiquen en la web del software: (8)

- libxml2 2.5.0
- libunwind
- PAPI
- MPI
- OpenMP

Algunas pueden ser satisfechas mediante repositorios:

```
# apt-get install libxml2 libxml2-dev  zlib1g-dev libtool
```

Para instalar libunwind debería ser así:

```
$ wget http://download.savannah.gnu.org/releases/libunwind/libunwind
  -1.1.tar.gz
$ tar xf libunwind-1.1.tar.gz
$ cd libunwind-1.1
$ CFLAGS="-O3 -mcpu=cortex-a15 -mtune=cortex-a15 -mfloat-abi=hard" \
./configure --prefix=/usr/local/libunwind
$ make -j4
# make install
```

PAPI es instalado de la siguiente manera:

```
$ wget http://icl.cs.utk.edu/projects/papi/downloads/papi-5.2.0.tar.
  gz
$ tar xf papi-5.2.0.tar.gz
$ cd papi-5.2.0/src
$ CFLAGS="-O3 -mcpu=cortex-a15 -mtune=cortex-a15 -mfloat-abi=hard" \
./configure --prefix=/opt/soft/papi
$ make -j4
# make install
```

Y la instalación de MPI la vemos en el apartado más adelante en detalle.

Y por tanto sólo quedaría instalar Extrae, pero surgieron algunos problemas:

- Aunque detecta libunwind durante el configure, cuando hace pruebas para comprobar que funciona durante el make, dice que no funciona, la solución fue desactivarlo mediante: –without-unwind
- No todas las versiones de MPI son compatibles, o eso parece, con Extrae, tuvimos que instalar la versión de OpenMPI 1.6 para conseguir que Extrae compilará
- La compilación pide mucha memoria, tanta que 2GB que tienen los nodos no es suficiente, hubo que añadir swap para que pudiese compilar y no acabar con un mensaje de *Out-of-memory*.

Una vez tenido esto en cuenta, las líneas para tener Extrae, una vez descargado de la web del BSC, son:

```
$ tar xf extrae-2.5.0.tar.gz
$ cd extrae-2.5.0
$ CFLAGS="-O3 -mcpu=cortex-a15 -mtune=cortex-a15 -mfloat-abi=hard -
  mfpu=neon" \
./configure --with-mpi=/usr/local/openmpi1.6 --enable-sampling --
  without-unwind \
--with-papi=/opt/soft/papi --enable-posix-clock --without-dyninst --
  prefix=/usr/local/extrae
$ make -j4
# make install
```

4.6. Message Passing Interface

En cuanto a version de MPI a instalar hemos instalado OpenMPI y MPICH, ya que si no contamos los distintos forks que tiene MPICH son los 2 grandes softwares de MPI. Así también podremos ver su rendimiento en nuestra red y arquitectura.

La instalación y configuración de ambos ha tenido poco contratiempos.

4.6.1. OpenMPI

Para instalas OpenMPI es muy sencillo:

```
$ wget http://www.open-mpi.org/software/ompi/v1.8/downloads/openmpi
  -1.8.1.tar.gz
$ tar xvf openmpi-1.8.1.tar.gz
$ cd openmpi-1.8.1
$ CFLAGS="-O3 -mcpu=cortex-a15 -mtune=cortex-a15 -mfloat-abi=hard" \
./configure --prefix=/usr/local/openmpi1.8.1
# make all install
```

Si después nos movemos a */usr/local openmpi1.8.1/bin* y ejecutamos:

```
$ mpicc -v
```

Vemos como se ha hecho el configure, y vemos que reconoce perfectamente la arquitectura ARM.

Para llamar a un programa que necesita de MPI con OpenMPI, hace falta hacerlo así:

```
$ /usr/local/openmpi1.8.1/bin/mpirun --hostfile hostfile -np #
  procesos /ruta/programa/mpi
```

Donde el fichero hostfile debe tener este aspecto:

```
usuario@192.168.0.81 slots=4
usuario@192.168.0.82 slots=4
usuario@192.168.0.83 slots=4
usuario@192.168.0.84 slots=4
usuario@192.168.0.85 slots=4
usuario@192.168.0.86 slots=4
```

Donde *usuario* debe poder hacer ssh sin contraseña, mediante clave pública que ya se hizo en la sección 4.3. Slot se refiere a cuántos procesadores tenemos disponibles en ese nodo.

4.6.2. MPICH

Para MPICH el procedimiento es casi idéntico e igual de sencillo:

```
$ wget http://www.mpich.org/static/downloads/3.1/mpich-3.1.tar.gz
$ tar xfz mpich-3.1.tar.gz
$ cd mpich-3.1/
$ CFLAGS="-O3 -mcpu=cortex-a15 -mtune=cortex-a15 -mfloat-abi=hard" \
 ./configure --prefix=/usr/local/mpich3.1
 MPICHLIB_CFLAGS="-O3 -mcpu=cortex-a15 -mtune=cortex-a15 -mfloat-abi
    =hard"
$ make
# make install
```

Para llamar a un programa que necesita de MPI con MPICH, hace falta hacerlo así:

```
$ /usr/local/mpich3.1/bin/mpirun -f hostfile -np #procesos /ruta/
   programa/mpi
```

Donde el fichero hostfile debe tener este aspecto:

```
192.168.0.81:4
192.168.0.82:4
192.168.0.83:4
192.168.0.84:4
192.168.0.85:4
192.168.0.86:4
```

Donde el usuario que ejecuta en local debe poder acceder por ssh mediante clave pública. El número después de la IP es cuantos procesadores hay disponibles en ese nodo.

4.6.3. Evaluación

Para evaluar las 2 instalaciones de MPI que tenemos usaremos unos benchmarks desarrollados por los mismos desarrolladores de MVAPICH, basado en MPICH. (29).

Estos benchmarks rinden métricas como el ancho de banda y la latencia de MPI.

Así que lo instalamos para OpenMPI:

```
$ wget http://mvapich.cse.ohio-state.edu/benchmarks/osu-micro-
   benchmarks-4.3.tar.gz
$ cd osu-micro-benchmarks-4.3
$ ./configure --prefix=/usr/local/osu_benchmarks_openmpi \
CC='/usr/local/openmpi1.8.1/bin/mpicc'
$ make
# make install
```

Y MPICH:

```
$ ./configure --prefix=/usr/local/osu_benchmarks_mpich
CC='/usr/local/mpich3.1/bin/mpicc'
$ make
# make install
```

Después, usamos los benchmarks de punto a punto, para saber cuanta latencia tenemos entre 2 nodos cualesquiera. El test en concreto mide la latencia de la siguiente manera:

- Un nodo envía un mensaje MPI de cierto tamaño a otro nodo y espera la respuesta
- El recibidor del mensaje, una vez recibido, le manda al primer nodo otro mensaje MPI del mismo tamaño

Para OpenMPI:

```
$ cd /usr/local/osu_benchmarks_openmpi/libexec/osu-micro-benchmarks/
    mpi/pt2pt/
$ /usr/local/openmpi1.8.1/bin/mpirun --hostfile ~/Hostfiles/pt2pt -
    np 2 osu_latency
```

Y MPICH:

```
$ cd /usr/local/osu_benchmarks_mpich/libexec/osu-micro-benchmarks/
    mpi/pt2pt/
$ /usr/local/mpich3.1/bin/mpirun -f ~/Hostfiles/pt2pt.mpich -np 2 ./
    osu_latency
```

Y obtenemos esta gráfica:

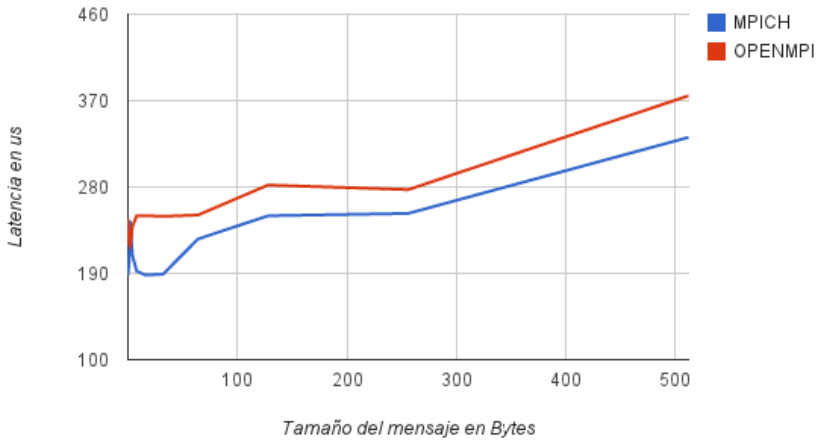

Figura 4.7: Prueba de latencia entre OpenMPI y MPICH

El comportamiento de ambos es parecido como se observa en 4.7, con fluctuaciones para tamaños muy pequeños, pero después se mantienen escalando linealmente al tamaño del paquete a enviar. Lo que sí vemos es que MPICH ofrece una menor latencia que OpenMPI.

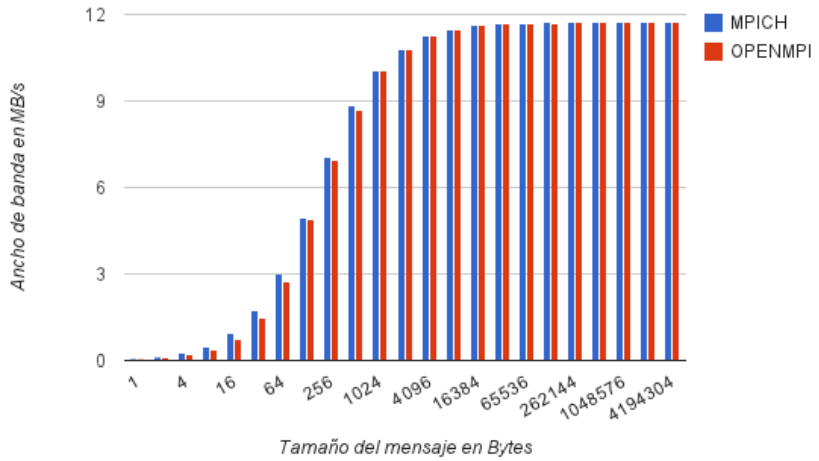

Figura 4.8: Prueba de ancho de banda entre OpenMPI y MPICH

En el gráfico 4.8 vemos que ambos tienden a aprovechar la totalidad del ancho de banda del que disponen, llegando casi a 12 MB/s, pequeña diferencia para tamaño de mensajes más pequeños, donde MPICH aprovecha mejor el ancho de banda que OpenMPI.

4.7. Librerías

En cuanto a librerías matemáticas necesarias para correr los distintos programas del SCC necesitamos ATLAS y FFTW, como no tenemos experiencia con las librerías básicas (instalación y configuración) optamos por usar éstas mismas y no usar suites como pueden ser las de Intel con su Intel math kernel library.

4.7.1. ATLAS

ATLAS es quizá el software que más esfuerzos ha necesitado para ser instalado, ya que se necesita aplicar un pequeño parche para poder correrlo sobre ARM y una ABI por hardware, además ATLAS hace una comprobación del sistema donde será instalado y la imagen del sistema operativo al no estar demasiado preparada para este tipo de entornos, hay pequeños datos que hemos tenido que darle a ATLAS.

Para que ATLAS se encargue de mejorar nuestra librería BLAS tenemos que hacer lo siguiente una vez hemos descargado el software de su página web: (37)

Aplicar el parche para nuestro sistema con ABI por hardware y ARM:

```
$ wget http://math-atlas.sourceforge.net/fixes/armhardfp_archdef.tar
$ tar xvf armhardfp_archdef.tar
```

Después es necesario editar los campos del fichero:
ATLAS/CONFIG/src/atlcomp.txt -mfloat-abi=softfp por -mfloat-abi=hard.

Seguidamente ya podemos instalar ATLAS, dentro de la carpeta de ATLAS:

```
$ mkdir build_ARM_a15
$ cd build_ARM_a15
$ CFLAGS="-O3 -mcpu=cortex-a15 -mtune=cortex-a15 -mfloat-abi=hard -
    mfpu=neon"
\../configure -m 800 -D c -DATL_ARM_HARDFP=1 -Ss ADdir /home/
    administrador/ARMHARDFP \
-t 4 -Fa alg -mcpu=cortex-a15 -mtune=cortex-a15 -mfloat-abi=hard -
    mfpu=neon \
--prefix=/usr/local/atlas
```

Donde los flags del configure hacen lo siguiente:

- **-m 800** Le indicamos que la frecuencia de nuestro procesador es 800MHz.
- **-D c -DATL_ARM_HARDFP=1 -Ss ADdir /home/administrador/ARMHARDFP** Sirve para aplicar el parche para ARM y ABI hard.
- **-t 4** Indicamos tenemos 4 procesadores (realmente 8, pero la imagen no tiene soporte para los 8), así que puede crear hasta 4 threads.
- **-Fa alg -mcpu=cortex-a15 -mtune=cortex-a15 -mfloat-abi=hard -mfpu=neon** Decimos que aplique estos flags de compilación a todos los compiladores que vaya a usar.
- **–prefix=/usr/local/atlas** Ruta donde queremos finalmente los ficheros de instalación

Por tanto seguimos con la instalación:

```
$ make build
```

Este paso puede llegar a tardar mucho, sobretodo sobre la plataforma que estamos compilando, en este paso es donde se analiza el hardware y sistema operativo para conseguir el mayor rendimiento. Cuidado, no compilar con varios threads *-j4*, ya que sino, después las posibilidades de que no se compile correctamente y fallen las comprobaciones posteriores son muy altas.

Después comprobamos que todo haya sido compilado correctamente:

```
$ make check
$ make ptcheck
```

Después podemos comprobar como de óptima es nuestra versión, comparándola con otras arquitecturas, compararemos con la que viene con el parche para:

```
$ ./xatlbench -dp /home/administrador/ARMHARDFP/ARMv732NEON/ -dc bin
    /INSTALL_LOG/
```

Y obtenemos estas tablas:

	Single precision			
	Real		**Complex**	
Benchmark	**Reference**	**Present**	**Reference**	**Present**
kSelMM	166.5	179.9	163.0	174.1
kGenMM	142.2	150.6	139.1	151.7
kMM_NT	139.1	123.4	131.9	137.9
BIG_MM	141.2	157.3	122.6	157.3
BIG_MM	171.8	175.5	170.6	172.0
kMV_N	16.0	75.0	50.8	83.7
kMV_T	30.9	74.7	58.8	102.3
kGER	20.4	60.0	47.2	92.0

Cuadro 4.2: Tiempos para precisión simple de ATLAS

	Double precision			
	Real		**Complex**	
Benchmark	**Reference**	**Present**	**Reference**	**Present**
kSelMM	86.4	182.6	95.4	176.4
kGenMM	68.9	103.9	71.8	132.8
kMM_NT	79.2	116.6	61.9	116.0
BIG_MM	64.0	130.4	67.6	103.9
BIG_MM	94.0	158.8	92.2	164.1
kMV_N	9.2	46.0	20.6	76.1
kMV_T	15.0	38.2	22.4	76.7
kGER	92.0	9.8	46.0	17.3

Cuadro 4.3: Tiempos para precisión doble de ATLAS

En las tablas, *reference* y *present* se refieren a la obtenida por la persona que creo los flags por defecto para la arquitectura y para la resultante compilada por uno mismo, respectivamente. Como vemos en los resultados de 4.2 y 4.3 nuestra versión compilada obtiene peor rendimiento que la original que compiló y parcheó el desarrollador del parche para ARM y ABI por hardware. Esto puede deberse a muchos factore como flags de compilación elegidos o la plataforma donde se obtuvieron los tiempos. Observamos también que nuestro rendimiento cae más con doble precisión que con simple, puede deberse a que ARMv7 (que usan nuestros procesadores) no tienen NEON (operaciones SIMD) de doble precisión.

Una vez tenemos todo comprobado sólo falta instalar mediante:

```
# make install
```

Una vez instalado hacemos una pequeña prueba para ver que nos ofrece mayor rendimiento, si nuestro ATLAS optimizado, si el ATLAS de repositorios de Ubuntu o simplemente no usar ATLAS y usar BLAS sin optimizar. Para ello usamos unos programas que justamente estresan este tipo de operaciones: (17)

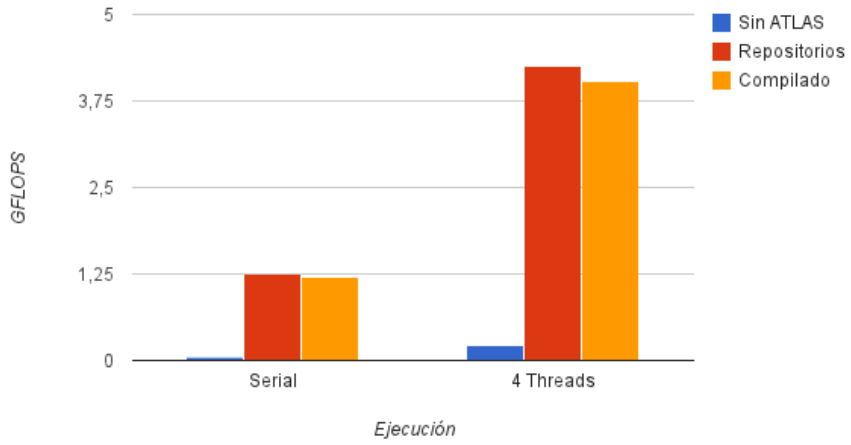

Figura 4.9: Prueba de rendimiento de ATLAS con doble precisión

En 4.9 vemos que la versión de repositorios obtiene unos mejores resultados que nuestra versión, aquí las razones no pueden ser muchas como en los tiempos de las tablas 4.2 y 4.3, ya que aquí seguramente se trata de optimizaciones a nivel de compilador más agresivas que las que hemos aplicado. Cabe decir que los resultados dados por todos los tests han sido correctos.

4.7.2. FFTW

Está librería es más sencilla de instalar que ATLAS, para ello tenemos que hacer:

```
$ wget http://www.fftw.org/fftw-3.3.4.tar.gz
$ tar zxvf fftw-3.3.4.tar.gz
$ CFLAGS="-O3 -mcpu=cortex-a15 -mtune=cortex-a15 -mfloat-abi=hard -
    mfpu=neon \
-I$OPENMPI/include" \
MPICC="$OPENMPI/bin/mpicc" \
LDFLAGS="-L$OPENMPI/lib/" \
MPIRUN="$OPENMPI/bin/mpirun" \
./configure --enable-mpi --enable-threads --prefix=/usr/local/fftw
    ARM_FLOAT_ABI=hardfp ARM_CPU_TYPE=cortex-a15
```

Lo que hacemos es descargar, descomprimir y hacer el configure de FFTW, aparte de los flags también le decimos donde está el compilador de MPI, sus librerías y cabeceras y con qué ejecutar programas MPI, en este caso lo enlazamos con la versión de OpenMPI instalada en 4.6.1

Los flags hacen lo siguiente:

- **–enable-mpi** Activamos MPI lo cuál hace que fftw compile también su librería para MPI y así ser capaz de hacer operaciones a través del todo clúster para minimizar el tiempo
- **–enable-threads** Activamos que pueda usar threads

- **–prefix=/usr/local/fftw** Donde queremos finalmente los ficheros de la instalación
- **ARM_FLOAT_ABI=hardfp ARM_CPU_TYPE=cortex-a15** Le decimos que nuestra ABI será por hardware y que nuestra CPU es un Cortex-a15

Después simplemente tenemos que acabar y realizar comprobaciones:

```
$ make
$ make bigcheck
# make install
```

Una vez tenemos funcionando la librería volvemos a realizar el mismo experimento que con ATLAS de comparar nuestra versión con la de repositorios y no usar FFTW. Los programas con el cuál probamos si no compilan con FFTW lo hacen por defecto con Fastest Fourier Transform in the East (FFTE).

Realizamos las pruebas para precisión simple y doble:

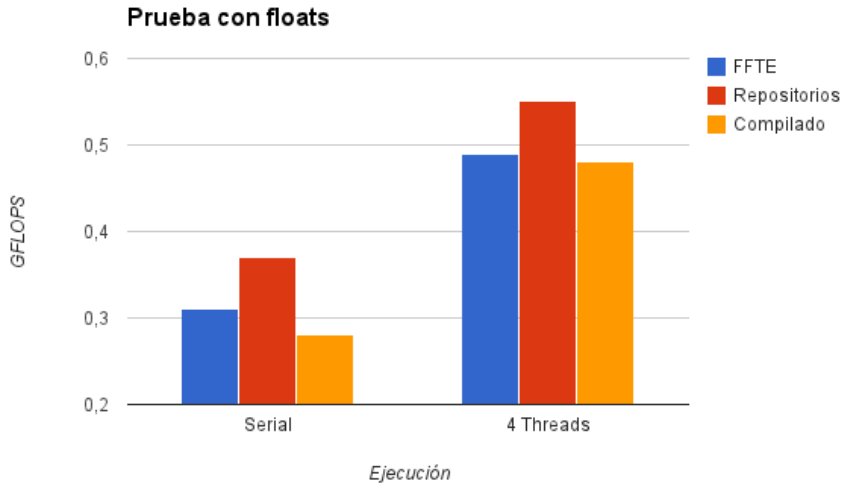

Figura 4.10: Prueba de rendimiento de FFTW con precisión simple

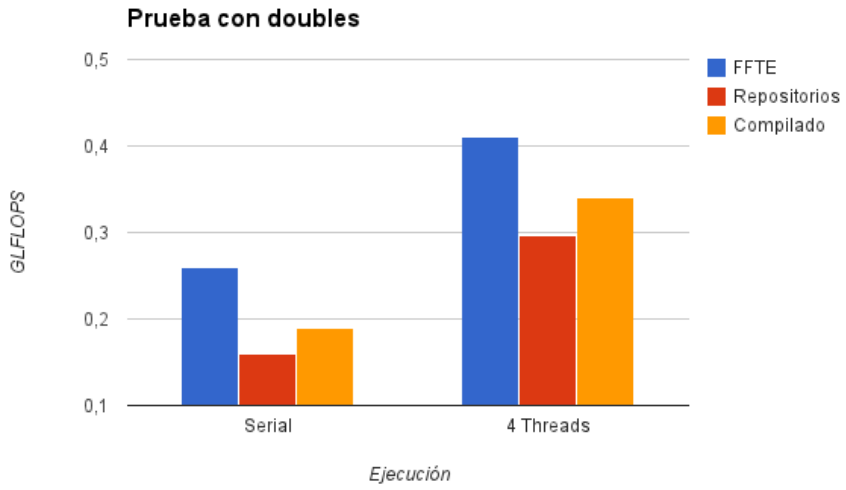

Figura 4.11: Prueba de rendimiento de FFTW con precisión doble

Vemos que en cuanto a precisión simple la versión de repositorios es la mejor de las 3 (FFTE, repositorios y nuestra versión compilada), pero en precisión doble vemos que FFTE queda como la que mejor rendimiento da, y en segundo lugar nuestra versión compilada.

El motivo de tales diferencias habría que buscarlas en las distintas optimizaciones al compilar la versión FFTE y la de repositorios e incluso comparar que FFTE no tenga más rendimiento que FFTW

4.8. Problemas con los nodos

Por supuesto, los nodos han presentado problemas que han dificultado todas las tareas realizadas, algunos no muy problemáticos pero que si afectan al rendimiento del clúster en general, y que vienen dados por el poco soporte que tienen a día de hoy las placas que estamos usando. Esto pequeños problemas son:

- Clock fijo a 800MHz.
- Problemas para botar por NFS.

Pero ha habido un error que sí que realmente ha afectado a la instalación y configuración del clúster y es que: su sistema de ficheros es altamente inestable, aún no sabemos los motivos porque sucede, pero el sistema de ficheros se corrompe aleatoriamente, esto hace que el sistema se ponga en modo de sólo lectura (esto es salvable ya que podríamos hacer que no lo hiciese, pero el sistema de ficheros ya estaría corrupto) y además suceden errores extraños, como por ejemplo que al intentar hacer un *make* nos diga que nuestro compilador (que hemos usado hace 5 minutos) no funciona, o no poder instalar/eliminar aplicaciones instaladas por repositorios por que sus ficheros están corruptos y no puede arreglar el error el gestor de paquetes, o también que al correr un simple script nos den *segmentation faults*.

Una primera solución a este error fue ejecutar los siguientes comandos:

```
# fsck.ext4 /dev/mmcblk1p3 -y
# reboot
```

En un primer momento parecía que funcionaba, pero no en todos los casos se arreglaban estos errores. Así que como última solución, optamos por el método rápido, y grabábamos imágenes del sistema operativo que estábamos usando de más. Y al menor síntoma de fallo de un nodo, se cambiaba la imagen.

Capítulo 5

Conclusiones

5.1. Conclusiones

Finalmente, repaso los objetivos que me propuse al principio del trabajo y doy mi conclusión personal al trabajo.

5.1.1. Objetivos

Los objetivos que tenía al principio de este trabajo y la valoración personal al final del mismo:

- Investigar el mundo de software de sistema que hay actualmente en HPC, a pesar de que recabar la información que se puede ver en el capítulo 3 ha sido más difícil de lo que pensaba en un primer momento, creo que se ve bien que tipo de software de sistema hay en el mercado a día de hoy.
- Seleccionar del todo software de sistema disponible un stack que nos valiese para el clúster, creo que también cumplido, aunque la solución elegida es lo que habitualmente se ve (cambiando pequeñas cosas)
- Instalar y configurar todo el software de sistema, opino que el sistema es bastante completo (a nivel experimental y no final), aunque el tiempo en este apartado ha jugado un papel más importante, puesto que no he podido tratar más con las librerías matemáticas para obtener un mejor rendimiento (ahora mismo instaladas desde repositorios u otra versión obtienen mejor rendimiento), ni con el MPI.

5.1.2. Conclusión personal

A nivel personal creo que, a pesar de que los resultados no son tan buenos como esperaba, la experiencia del trabajo ha sido muy buena, he aprendido un montón, lo básico en este mundo creo. Aunque ha habido una parte de investigación a la que se ha dedicado tiempo, realmente en la parte práctica es donde te enfrentas a más problemas que no siempre tienen una solución a corto plazo, y necesitan más tiempo, y esto es quizá algo que me hubiese gustado más, tener más tiempo para poder tocar y arreglar distintas cosas que creo que podrían estar mejor.

5.2. Trabajo futuro

Como trabajo futuro quedan muchas cosas:

- Conseguir mayor rendimiento en las librerías matemáticas instaladas, compilar con distintos flags, distintos compiladores... En definitiva, probar cómo podemos obtener mejor rendimiento compilando cosas.
- Mejorar las distintas versiones de MPI que hay en el sistema y probar otras versiones, y ver con cual obtenemos mejor rendimiento.
- Instalar un sistema de colas para gestionar los posibles usuarios que usen el clúster.
- Instalar un sistema de monitorización como Nagios para comprobar que ningún nodo esté caído.
- Crear una infraestructura de sistema de ficheros mejor, compartir los homes de los nodos, compartir /usr/local para tener nodos unificados y no tener que tratar con imágenes que actualizamos cada cierto tiempo. Este paso viene dado por una condición, y es mejorar el sistema de interconexión, que ahora mismo no obtenemos un buen rendimiento cuando tratamos con red.
- Mejorar la imagen que usamos, ésta última es quizá la más difícil pero la más interesante, mejorar el soporte para distintas cosas como frecuencia dinámica, poder botar por NFS para evitar el sistema de imágenes con la SD, incluso tocar cosas del kernel para optimizarlo para propósitos de HPC.

Capítulo 6

Conclusiones de equipo

6.1. Conclusiones de cara a la competición

En el **estado actual** de nuestro proyecto resulta inviable participar en la competición por varios motivos.

No reboot - Always Up. La normativa de la competición establece que las máquinas no se pueden reiniciar ni apagar durante toda la competición (excepto por motivos de seguridad). Nuestro sistema es todavía inestable, hasta el punto que resulta improbable sobrevivir cinco días sin reiniciar o que alguno de los nodos falle, de hecho sería improbable aguantar uno. Además se prevé tener que instalar nuevo software e incluso alguna librería dado que hay un set de aplicaciones a ejecutar que se darán a conocer durante la competición. Sin duda, este sería un objetivo prioritario y una manera de abordar este problema sería abandonar el uso de tarjetas SD hacía un sistema de inicio y ficheros en red mediante NFS.

Rendimiento. El rendimiento no es bueno, al menos no suficiente. En ediciones pasadas de la competición se alcanzaron los 8.5 TFLOPS en HPL que, en el peor caso, suponen 2.8 GFLOPS/W, 12 veces más que nuestros 0.229 GFLOPS/W actuales. No sería realista esperar asistir por primera vez con una máquina preparada para competir por los primeros puestos, pero si que podamos competir por no quedar últimos. Consideramos que una primera meta sería desarrollar un prototipo con unas previsiones de rendimiento mínimas de 6 TFLOPS en HPL para plantear seriamente el asistir.

Preparación, tanto de los componentes del equipo como del clúster. Aún teniendo la formación adecuada, tomar decisiones de diseño, montar una máquina y prepararla a nivel de software de sistema y aplicaciones requiere mucho tiempo. Contar que además el equipo debería ser de seis personas, por un lado agiliza el desarrollo pero por otro lado requiere mayor nivel de coordinación. Además entendemos que, para formar un equipo estable, se requeriría una persona vinculada a la FIB con conocimiento de primera mano del estado del arte HPC. Asumiría el rol de director equipo y además de coordinar debería garantizar la continuidad de la representación de la facultad en sucesivas ediciones.

Recursos. Este proyecto ha podido llevarse a cabo únicamente con recursos cedidos por el departamento de AC y por el BSC. Creemos que el desarrollo de un prototipo competitivo a pequeña escala podría hacerse con un coste no mucho mayor, pero en el caso de asistir a la

competición, haría falta un desembolso de dinero considerable. Es en este punto en el que entran en juego los sponsors. Todos los equipos que asisten a la competición están esponsorizados por grandes fabricantes de hardware que aprovechan la competición a modo de escaparate. Nuestro plan sería conseguir uno o varios sponsors que financiaran como mínimo, el coste total de los componentes del clúster y el desplazamiento y alojamiento durante el evento.

6.2. Trabajo futuro

Este trabajo, pese a demostrar no estar preparados para la competición, ha servido para sentar las bases a partir de las cuales poder seguir trabajando y mejorar los puntos débiles de nuestro prototipo.

De cara al futuro, la primera acción a realizar sería valorar todas las opciones y decidir si continuamos desarrollando un clúster basado en tecnología móvil, si se decide abandonar en favor de tecnología HPC más habitual (p.ej: Intel Xeon y aceleradores Nvidia) o valorar otras opciones menos estudiadas y no mencionadas como por ejemplo: procesadores móviles con aceleradores externos Nvidia.

Personalmente creemos que en el ámbito de procesadores móviles queda mucho trabajo por hacer y que lo mejor está por venir. En esta línea, por tanto, tenemos mucho trabajo futuro sobre la mesa. A corto plazo empezaríamos por abordar el problema de la red de interconexión de nodos, y a la vez, trataríamos de hacer un mejor uso del hardware disponible: aceleradores, ocho núcleos y máxima frecuencia de procesador entre otros. A medio plazo se buscaría obtener nuevas placas de desarrollo con arquitectura ARMv8 de 64-bits y profundizar en el análisis y optimización profunda de las aplicaciones.

Agradecimientos

Por último y no por ello menos importante, me gustaría agradecer a ciertas personas la ayuda brindada para poder acabar este proyecto:

A Álex Ramírez, por darnos la idea para el trabajo, por ayudarnos en temas técnicos y proporcionarnos el material que hemos usado. Sin él seguramente este proyecto ni habría nacido.

A David López, por brindar la ayuda necesaria como director para poder completar el trabajo, dando indicaciones cuando eran necesarias.

A mi compañero de carrera Uri, por ayudarnos con temas que no habrían acabado tan bien sin él.

Y para finalizar, como no, a mis compañeros Constan y David, por hacer más llevadero algo tan importante como es un proyecto de final de grado.

Apéndices

Apéndice A

Script: cambiar hostname

```
#!/bin/bash
#Script para cambiar el hostname de un nodo
#por el introducido por nosotros
hostname=$(cat /etc/hostname)

echo $hostname
echo "Plaquita's number"
read number

if [ $number = "81" ]; then
  newhostname="81-Sven"
elif [ $number = "82" ]; then
  newhostname="82-Tresdin"
elif [ $number = "83" ]; then
  newhostname="83-Furion"
elif [ $number = "84" ]; then
  newhostname="84-Rylai"
elif [ $number = "85" ]; then
  newhostname="85-Mirana"
elif [ $number = "86" ]; then
  newhostname="86-Lina"
fi

echo $newhostname

#Modify files that includes hostname
sudo sed -i "s/$hostname/$newhostname/" /etc/hosts
sudo sed -i "s/$hostname/$newhostname/" /etc/hostname

sudo reboot
```

Apéndice B

Script: copiar clave pública

```
#!/bin/bash
#Genera llave publica y la copia a toda la red de nodos
ssh-keygen
for i in {81..86}
do
  ssh-copy-id 192.168.0.$i
done
```

Apéndice C

Script: ssh a nodos

```
#!/bin/bash
# Script para acceder rapidamente a los nodos
#

ip="192.168.0."

if [ $# = "2" ]; then
  echo "Bad arguments"
  exit -1;
fi;

if [ $1 = "81" ] || [ $1 = "sven" ]; then
  ip+="81"
elif [ $1 = "82" ] || [ $1 = "tresdin" ]; then
  ip+="82"
elif [ $1 = "83" ] || [ $1 = "furion" ]; then
  ip+="83"
elif [ $1 = "84" ] || [ $1 = "rylai" ]; then
  ip+="84"
elif [ $1 = "85" ] || [ $1 = "mirana" ]; then
  ip+="85"
elif [ $1 = "86" ] || [ $1 = "lina" ]; then
  ip+="86"
else
  echo "That plaquita no esta"
  exit -1;
fi
  ssh administrador@$ip

exit 0
```

Abreviaciones

ABI Application binary interface. 46, 51–53, 55

ATLAS Automatically Tuned Linear Algebra Software. 7, 33–35

BSC Barcelona Supercomputing Center. 18, 30, 47

CNK Compute Node Kernel. 23

CPU Central Processing Unit. 36, 41, 44, 55

eMMC Embedded Multi-Media Card. 36

FFTE Fastest Fourier Transform in the East. 55, 56

FFTW Fastest Fourier Transform in the West. 7, 23, 33, 35, 55, 56

GB Gigabyte. 36, 47

GDB GNU Debugger. 28

GPU Graphics Processing Unit. 36

HPC High Perfomance Computing. 2, 8, 11–14, 18, 22, 27, 33, 58, 59

HPL High-Performance Linpack. 11

ISA Instruction Set Architecture o en castellano conjunto de instrucciones. 28

ISC International Supercomputing Conference. 8, 10, 11

Mbps Mega Bits Por Segundo. 36

MHz Megahercios. 36, 52, 56

MPI Message Passing Interface. 28, 30–33, 41, 47–50, 54, 58, 59

NFS Network File System. 41, 42, 56, 59

SCC Student Cluster Competition. 8, 10–12, 14, 28, 33, 51

SD Secure Digital. 36, 39, 40, 59

SIMD Single Instruction, Multiple Data o en castellano una instrucción, múltiples datos. 28, 53

SO Sistema operativo. 7, 22, 23, 25

SoC System On Chip. 36, 39, 40

W Vatios. 10, 11

Glosario

Blue Gene Família de supercomputadores fabricados por IBM. 23, 25, 30

Compute Unified Device Architecture Modelo de programación creado por Nvidia para la programación de sus GPUs. 30

dd Comando de Linux que permite copiar información a bajo nivel. 39

GNU Compiler Collection Colección de compiladores creados por el proyecto GNU. 28

Intel compiler collection Compiladores de C, C++ y fortran desarollados por Intel. 7, 22, 25, 29

Intel math kernel library Librería matemática desarollada por Intel. 22, 25, 33–35, 51

Linux Sistema operativo. 26, 28, 39

MASS Mathematical Acceleration Subsystem Libraries de IBM. 23

Netlib Repositorio de software para computación científica, incluye distintas librerías como LAPACK o LINPACK. 23

Open64 Compiler Suite Compiladores de C, C++ y fortran que desarrolla en parte AMD. 30

TOP500 Web que recoge datos de los 500 supercomputadores más potentes del mundo.. 7, 22, 25, 26

Xeon Phi Acelerador de Intel, corre Linux. 25

Bibliografía

[1] Ayesha .A. Why do super computers use linux?, 2012. URL http://www.unixmen.com/why-do-super-computers-use-linux/. [Online; accedido en 10-Mayo-2014].

[2] Inc. Adaptive Computing. Torque resource manager - adaptive computing, 2014. URL http://www.adaptivecomputing.com/products/open-source/torque/. [Online; accedido en 8-Junio-2014].

[3] AMD. Acml – amd core math library, 2014. URL http://developer.amd.com/tools-and-sdks/cpu-development/cpu-libraries/amd-core-math-library-acml/. [Online; accedido en 11-Junio-2014].

[4] AMD. x86 open64 compiler suite, 2014. URL http://developer.amd.com/tools-and-sdks/cpu-development/cpu-tools-sdks/x86-open64-compiler-suite/. [Online; accedido en 8-Junio-2014].

[5] AMD. Acml – amd core math library, 2014. URL http://www.math.nsc.ru/conference/sobolev/100/PDF/Subsection_ECMP/Kostin.pdf. [Online; accedido en 11-Junio-2014].

[6] ARM. Arm ds-5 development studio, 2014. URL http://ds.arm.com/. [Online; accedido en 8-Junio-2014].

[7] Blaise Barney. Using the sequoia and vulcan bg/q systems, 2014. URL https://computing.llnl.gov/tutorials/bgq/#Software. [Online; accedido en 25-Abril-2014].

[8] BSC. Extrae, 2014. URL http://www.bsc.es/computer-sciences/extrae. [Online; accedido en 28-Junio-2014].

[9] Computerworld. Ibm's blue gene supercomputers to run linux, 2014. URL http://www.computerworld.com/s/article/75384/IBM_s_Blue_Gene_supercomputers_to_run_Linux. [Online; accedido en 17-Abril-2014].

[10] HPC Advisory Council. Hpc advisory council - isc'14 student cluster competition - competition rules, 2014. URL http://www.hpcadvisorycouncil.com/events/2014/isc14-student-cluster-competition/index.php. [Online; accedido en 17-Abril-2014].

[11] Federico D. Sacerdoti-Mason J. Katz-Matthew L. Massie-David E. Culler. Wide area cluster monitoring with ganglia. 2003. [Disponible en: http://beta.ele.uva.es/rocks-documentation/4.3/papers/cluster2003-ganglia.pdf].

[12] Jack Dongarra. Visit to the national university for defense technology changsha, china. 2013. [Disponible en: http://www.netlib.org/utk/people/JackDongarra/PAPERS/tianhe-2-dongarra-report.pdf].

[13] Nagios Enterprises. Nagios - the industry standard in it infrastructure monitoring, 2014. URL http://www.nagios.org/. [Online; accedido en 31-Mayo-2014].

[14] Swedish National Infrastructure for Computing. Nsc - snic, 2014. URL http://www.snic.se/apply-for-resources/available-resources/nsc. [Online; accedido en 25-Abril-2014].

[15] Matteo Frigo and Steven G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005. Special issue on "Program Generation, Optimization, and Platform Adaptation".

[16] The Cacti Group. Cacti - the complete rrdtool-based graphing solution, 2014. URL http://www.cacti.net/. [Online; accedido en 31-Mayo-2014].

[17] Constantino Gómez. Diseño y evaluación de un clúster hpc: aplicaciones, 2014.

[18] Hewlett-Packard. Cluster services – unified cluster portfolio, 2014. URL http://h20311.www2.hp.com/HPC/cache/275420-0-0-0-121.html. [Online; accedido en 25-Abril-2014].

[19] IBM. Blue gene, 2014. URL http://www-03.ibm.com/ibm/history/ibm100/us/en/icons/bluegene/breakthroughs/. [Online; accedido en 17-Abril-2014].

[20] IBM. Ibm - software - c and c++ compilers family, 2014. URL http://www-03.ibm.com/software/products/en/ccompfami. [Online; accedido en 8-Junio-2014].

[21] Cray Inc. Cray cs300 series cluster supercomputers: Hpc cluster software stack, 2014. URL http://www.cray.com/Products/Computing/CS/Software/HPCClusterSoftwareStack.aspx. [Online; accedido en 25-Abril-2014].

[22] Intel. Optimization notice, 2014. URL https://software.intel.com/en-us/articles/optimization-notice. [Online; accedido en 8-Junio-2014].

[23] Intel. Intel® composer xe suites, 2014. URL https://software.intel.com/en-us/intel-composer-xe/. [Online; accedido en 2-Junio-2014].

[24] Intel. Intel math kernel library, 2014. URL https://software.intel.com/en-us/intel-mkl. [Online; accedido en 11-Junio-2014].

[25] Argonne National Laboratory. Mpich, high-performance portable mpi, 2014. URL http://www.mpich.org/. [Online; accedido en 11-Mayo-2014].

[26] Argonne National Laboratory. Mpich2, perfomance and portability, 2014. URL http://www.cels.anl.gov/events/conferences/SC07/presentations/mpich2-flyer.pdf. [Online; accedido en 11-Mayo-2014].

[27] Linaro. Linaro: open source software for arm socs, 2014. URL http://www.linaro.org/. [Online; accedido en 15-Abril-2014].

[28] SchedMD LLC. Simple linux utility for resource management, 2014. URL http://www.schedmd.com/slurmdocs/slurm.html. [Online; accedido en 8-Junio-2014].

[29] NBCL. Benchmarks — network-based computing laboratory, 2014. URL http://mvapich.cse.ohio-state.edu/benchmarks/. [Online; accedido en 30-Junio-2014].

[30] Tobi Oetiker. Mrtg - tobi oetiker's mrtg - the multi router traffic grapher, 2014. URL http://oss.oetiker.ch/mrtg/. [Online; accedido en 31-Mayo-2014].

[31] The Ganglia Project. Ganglia monitoring system, 2014. URL http://ganglia.sourceforge.net/. [Online; accedido en 31-Mayo-2014].

[32] The Open MPI Project. Open mpi: Open source high performance computing, 2014. URL http://www.open-mpi.org. [Online; accedido en 11-Mayo-2014].

[33] Zabbix SIA. Homepage of zabbix :: An enterprise-class open source distributed monitoring solution, 2014. URL http://www.zabbix.com/. [Online; accedido en 31-Mayo-2014].

[34] Matthew J. Sottile and Ronald G. Minnich. Supermon: A high-speed cluster monitoring system. In *In Proc. of IEEE Intl. Conference on Cluster Computing*, pages 39–46, 2002.

[35] TOP500. Top500, 2014. URL http://www.top500.org. [Online; accedido en 7-Abril-2014].

[36] David Trilla. Diseño y evaluación de un clúster hpc: hardware, 2014.

[37] R. Clint Whaley, Antoine Petitet, and Jack J. Dongarra. Automated empirical optimization of software and the ATLAS project. *Parallel Computing*, 27(1–2):3–35, 2001. Web: http://math-atlas.sourceforge.net/.

[38] Wikipedia. Message passing interface, 2014. URL http://en.wikipedia.org/wiki/Message_Passing_Interface#Example_program. [Online; accedido en 11-Mayo-2014].